Глава 29. Линейное программирование

Многие задачи можно сформулировать как задачи максимизации или минимизации некой цели в условиях ограниченности ресурсов и наличия конкурирующих ограничений. Если удастся задать цель в виде линейной функции нескольких переменных и сформулировать ограничения в виде равенств или неравенств, связывающих эти переменные, можно получить ***задачу линейного программирования*** (linear-programming problem). Задачи линейного программирования часто встречаются в разнообразных практических приложениях. Начнем их изучение на примере подготовки предвыборной кампании.

Политическая задача

Представьте себя на месте политика, стремящегося выиграть выборы. В вашем избирательном округе есть районы трех типов — городские, пригородные и сельские. В этих районах зарегистрировано соответственно 100, 200 и 50 тысяч избирателей. Чтобы добиться успеха, желательно получить большинство голосов в каждом из трех регионов. Вы честный человек и никогда не будете давать обещания, в которые сами не верите. Однако вам известно, что определенные пункты программы могут помочь завоевать голоса тех или иных групп избирателей. Основными пунктами программы являются строительство дорог, контроль над использованием огнестрельного оружия, сельскохозяйственные субсидии и налог на бензин, направляемый на улучшение работы общественного транспорта. Согласно исследованиям вашего предвыборного штаба можно оценить, сколько голосов будет приобретено или потеряно в каждой группе избирателей, если потратить тысячу долларов на пропаганду по каждому пункту программы. Эта информация представлена в таблице на рис. 29.1. Каждый элемент данной таблицы показывает, сколько тысяч избирателей из городских, пригородных и сельских районов удастся привлечь, потратив тысячу долларов на агитацию в поддержку определенного пункта предвыборной программы. Отрицательные элементы отражают потерю голосов. Задача состоит в минимизации суммы, которая позволит завоевать 50 тысяч голосов горожан, 100 тысяч голосов избирателей из пригородных зон и 25 тысяч голосов сельских жителей.

Методом проб и ошибок можно выработать стратегию, которая позволит получить необходимое количество голосов, но затраты на такую стратегию могут оказаться не самыми низкими. Например, можно выделить 20 тысяч долларов на пропаганду строительства дорог, 0 долларов на агитацию в пользу контроля над

Пункт программы	Горожане	Жители пригорода	Сельские жители
Строительство дорог	−2	5	3
Контроль над оружием	8	2	−5
Сельскохозяйственные субсидии	0	0	10
Налог на бензин	10	0	−2

Рис. 29.1. Влияние предвыборной политики на настроения избирателей. Каждая запись представляет собой количество тысяч голосов горожан, жителей пригорода и сельских жителей, получаемых при затрате тысячи долларов на рекламу определенных пунктов предвыборной программы. Отрицательные элементы отражают потерю голосов.

использованием оружия, 4 тысячи долларов на пропаганду сельскохозяйственных субсидий и 9 тысяч долларов на агитацию за введение налога на бензин. При этом удастся привлечь $20(-2) + 0(8) + 4(0) + 9(10) = 50$ тысяч голосов горожан, $20(5) + 0(2) + 4(0) + 9(0) = 100$ тысяч голосов жителей пригородов и $20(3) + 0(-5) + 4(10) + 9(-2) = 82$ тысячи голосов сельских жителей. Таким образом, будет получено ровно необходимое количество голосов в городских и пригородных районах и большее количество голосов в сельской местности. (Получается, что в сельской местности привлечено голосов больше, чем имеется избирателей!) Чтобы получить эти голоса, придется потратить на пропаганду $20 + 0 + 4 + 9 = 33$ тысячи долларов.

Возникает естественный вопрос: является ли данная стратегия наилучшей из возможных, т.е. можно ли достичь поставленных целей, потратив на пропаганду меньше денег? Ответ на этот вопрос можно получить, продолжая действовать методом проб и ошибок, однако вместо этого хотелось бы иметь некий систематический метод для ответа на подобные вопросы. Сформулируем данный вопрос математически. Введем четыре переменные:

- x_1 — сумма (в тысячах долларов), потраченная на пропаганду программы строительства дорог;
- x_2 — сумма (в тысячах долларов), потраченная на агитацию в пользу контроля над использованием оружия;
- x_3 — сумма (в тысячах долларов), потраченная на пропаганду программы сельскохозяйственных субсидий;
- x_4 — сумма (в тысячах долларов), потраченная на агитацию за введение налога на бензин.

Требование получить не менее 50 тысяч голосов избирателей-горожан можно записать в виде неравенства

$$-2x_1 + 8x_2 + 0x_3 + 10x_4 \geq 50 . \qquad (29.1)$$

Аналогично требования получить не менее 100 тысяч голосов избирателей, живущих в пригороде, и 25 тысяч голосов избирателей в сельской местности можно

записать как неравенства

$$5x_1 + 2x_2 + 0x_3 + 0x_4 \geq 100 \tag{29.2}$$

и

$$3x_1 - 5x_2 + 10x_3 - 2x_4 \geq 25 \ . \tag{29.3}$$

Любые значения переменных x_1, x_2, x_3, x_4, удовлетворяющие неравенствам (29.1)–(29.3), позволят получить достаточное количество голосов избирателей в каждом регионе. Чтобы сделать затраты минимально возможными, необходимо минимизировать сумму, затраченную на пропаганду, т.е. минимизировать выражение

$$x_1 + x_2 + x_3 + x_4 \ . \tag{29.4}$$

Хотя отрицательная агитация часто встречается в политической борьбе, отрицательных затрат на пропаганду не бывает. Поэтому следует записать условия

$$x_1 \geq 0, \ x_2 \geq 0, \ x_3 \geq 0 \ \text{ и } \ x_4 \geq 0 \ . \tag{29.5}$$

Объединив неравенства (29.1)–(29.3) и (29.5) для минимизации (29.4), мы получаем то, что известно как задача линейного программирования. Запишем ее следующим образом:

минимизировать
$$x_1 + x_2 + x_3 + x_4 \tag{29.6}$$

при условиях
$$-2x_1 + 8x_2 + 0x_3 + 10x_4 \geq 50 \tag{29.7}$$
$$5x_1 + 2x_2 + 0x_3 + 0x_4 \geq 100 \tag{29.8}$$
$$3x_1 - 5x_2 + 10x_3 - 2x_4 \geq 25 \tag{29.9}$$
$$x_1, x_2, x_3, x_4 \geq 0. \tag{29.10}$$

Решение этой задачи линейного программирования позволит политику получить оптимальную стратегию предвыборной агитации.

Общий вид задач линейного программирования

В общем случае в задаче линейного программирования требуется оптимизировать некую линейную функцию при условии выполнения множества линейных неравенств. Для данных действительных чисел $a_1, a_2, \ldots, a_n$ и множества переменных $x_1, x_2, \ldots, x_n$ ***линейная функция*** этих переменных f определяется как

$$f(x_1, x_2, \ldots, x_n) = a_1x_1 + a_2x_2 + \cdots + a_nx_n = \sum_{j=1}^{n} a_jx_j \ .$$

Если b представляет собой действительное число, а f является линейной функцией, то уравнение

$$f(x_1, x_2, \ldots, x_n) = b$$

называется ***линейным равенством***, а неравенства

$$f(x_1, x_2, \ldots, x_n) \leq b$$

и

$$f(x_1, x_2, \ldots, x_n) \geq b$$

называются линейными неравенствами. Термин ***линейные ограничения*** применяется как к линейным равенствам, так и к линейным неравенствам. В линейном программировании не допускается использование строгих неравенств. Формально ***задача линейного программирования*** является задачей минимизации или максимизации линейной функции при соблюдении конечного множества линейных ограничений. Если выполняется минимизация, то такая задача называется ***задачей минимизации***, а если выполняется максимизация, то такая задача называется ***задачей максимизации***.

Вся оставшаяся часть данной главы будет посвящена формулированию и решению задач линейного программирования. Для решения задач линейного программирования существует несколько алгоритмов с полиномиальным временем выполнения, однако изучать их в данной главе мы не будем. Вместо этого мы рассмотрим самый старый алгоритм линейного программирования — симплекс-алгоритм. В наихудшем случае симплекс-алгоритм не выполняется за полиномиальное время, однако обычно он достаточно эффективен и широко используется на практике.

Обзор задач линейного программирования

Чтобы описывать свойства задач линейного программирования и алгоритмы их решения, удобно договориться, в каких формах их записывать. В данной главе мы будем использовать две формы: ***стандартную*** и ***каноническую*** (slack). Их строгое определение будет дано в разделе 29.1. Неформально стандартная форма задачи линейного программирования представляет собой задачу максимизации линейной функции при соблюдении линейных *неравенств*, в то время как каноническая форма является задачей максимизации линейной функции при соблюдении линейных *равенств*. Обычно мы будем использовать стандартную форму задач линейного программирования, однако при описании принципа работы симплекс-алгоритма удобнее использовать каноническую форму. На данном этапе ограничимся рассмотрением задачи максимизации линейной функции от n переменных при условии выполнения m линейных неравенств.

Начнем с рассмотрения следующей задачи линейного программирования с двумя переменными:

максимизировать
$$x_1 + x_2 \tag{29.11}$$
при условиях
$$4x_1 - x_2 \leq 8 \tag{29.12}$$
$$2x_1 + x_2 \leq 10 \tag{29.13}$$
$$5x_1 - 2x_2 \geq -2 \tag{29.14}$$
$$x_1, x_2 \geq 0. \tag{29.15}$$

Любой набор значений переменных x_1 и x_2, удовлетворяющий всем ограничениям (29.12)–(29.15), называется ***допустимым решением*** (feasible solution) дан-

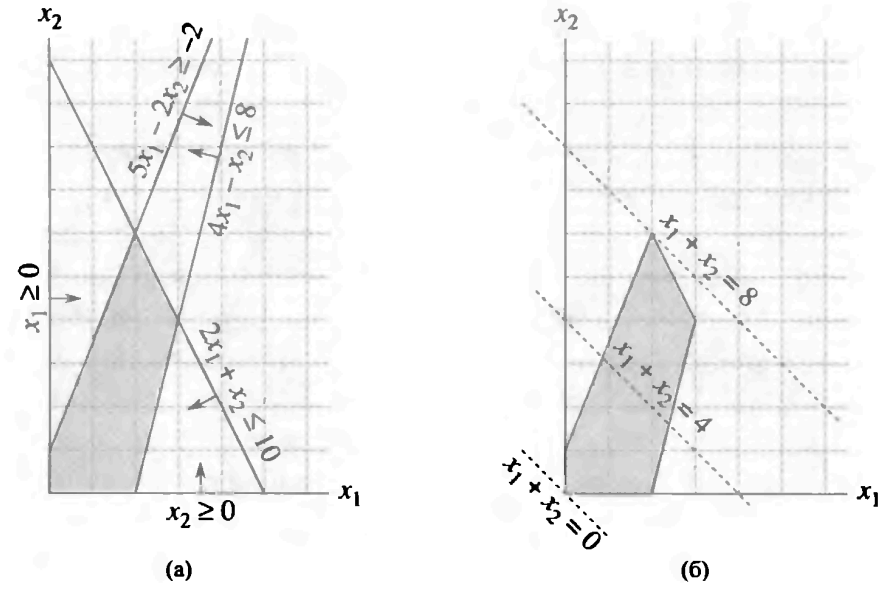

Рис. 29.2. **(а)** Задача линейного программирования (29.12)–(29.15). Каждое ограничение представлено линией и направлением. Пересечение ограничений, представляющее область допустимых решений, заштриховано. **(б)** Пунктирными линиями показаны точки, для которых целевое значение равно соответственно 0, 4 и 8. Оптимальным решением данной задачи линейного программирования является $x_1 = 2$ и $x_2 = 6$ с целевым значением 8.

ной задачи линейного программирования. Если изобразить эти ограничения в декартовой системе координат (x_1, x_2), как показано на рис. 29.2, (а), то множество допустимых решений (заштрихованная область на рисунке) образует выпуклую область[1] в двумерном пространстве. Эта выпуклая область называется ***областью допустимых решений***, или допустимой областью (feasible region). Функция, которую мы хотим максимизировать, называется ***целевой функцией*** (objective function). Теоретически можно было бы оценить значение целевой функции $x_1 + x_2$ в каждой точке допустимой области (значение целевой функции в определенной точке называется ***целевым значением*** (objective value)). Затем можно найти точку, в которой целевое значение максимально; она и будет оптимальным решением. В данном примере (как и в большинстве задач линейного программирования) допустимая область содержит бесконечное множество точек, поэтому хотелось бы найти способ, который позволит находить точку с максимальным целевым значением, не прибегая к вычислению значений целевой функции в каждой точке допустимой области.

В двумерном случае оптимальное решение можно найти с помощью графической процедуры. Множество точек, для которых $x_1 + x_2 = z$, при любом z представляет собой прямую с коэффициентом наклона -1. Если мы построим график функции $x_1 + x_2 = 0$, получится прямая с коэффициентом наклона -1, проходящая через начало координат, как показано на рис. 29.2, (б). Пересечение

[1] Интуитивно выпуклая область определяется как область, удовлетворяющая тому требованию, что для любых двух точек, принадлежащих области, все точки соединяющего их отрезка также должны принадлежать этой области.

данной прямой и допустимой области представляет собой множество допустимых решений, целевое значение в которых равно 0. В данном случае пересечением прямой и допустимой областей является точка $(0, 0)$. В общем случае для любого z пересечением прямой $x_1 + x_2 = z$ с допустимой областью является множество допустимых решений, в которых целевое значение равно z. На рис. 29.2, (б) показаны прямые $x_1 + x_2 = 0$, $x_1 + x_2 = 4$ и $x_1 + x_2 = 8$. Поскольку допустимая область на рис. 29.2 ограничена, должно существовать некоторое максимальное значение z, для которого пересечение прямой $x_1 + x_2 = z$ и допустимой области является непустым множеством. Любая точка этого пересечения является оптимальным решением задачи линейного программирования; в данном случае такой точкой является $x_1 = 2$, $x_2 = 6$ с целевым значением 8.

То, что оптимальное решение задачи линейного программирования оказалось в некоторой вершине допустимой области, не случайно. Максимальное значение z, при котором прямая $x_1 + x_2 = z$ пересекает допустимую область, должно находиться на границе допустимой области, поэтому пересечение данной прямой и границы допустимой области может быть либо вершиной, либо отрезком. Если пересечение является вершиной, то существует единственное оптимальное решение, находящееся в данной вершине. Если же пересечение является отрезком, то все точки этого отрезка имеют одинаковое целевое значение и являются оптимальными решениями; в частности, оптимальными решениями являются оба конца отрезка. Каждый конец отрезка является вершиной, поэтому в данном случае также существует оптимальное решение в вершине допустимой области.

Несмотря на то что для задач линейного программирования, в которых число переменных больше двух, простое графическое решение построить невозможно, наши интуитивные соображения остаются в силе. В случае трех переменных каждое ограничение описывается полупространством в трехмерном пространстве. Пересечение этих полупространств образует допустимую область. Множество точек, в которых целевая функция имеет значение z, теперь представляет собой некоторую плоскость. Если все коэффициенты целевой функции неотрицательны и начало координат является допустимым решением рассматриваемой задачи линейного программирования, то при движении этой плоскости по направлению от начала координат получаются точки с возрастающими значениями целевой функции. (Если начало координат не является допустимым решением или некоторые коэффициенты целевой функции отрицательны, интуитивная картина будет немного сложнее.) Как и в двумерном случае, поскольку допустимая область выпукла, множество точек, в которых достигается оптимальное целевое значение, должно содержать вершину допустимой области. Аналогично в случае n переменных каждое ограничение определяет полупространство в n-мерном пространстве. Допустимая область, образуемая пересечением этих полупространств, называется ***симплексом*** (simplex). Целевая функция в этом случае представляет собой гиперплоскость, и благодаря выпуклости допустимой области оптимальное решение находится в некоторой вершине симплекса.

Симплекс-алгоритм получает на вход задачу линейного программирования и возвращает оптимальное решение. Он начинает работу в некоторой вершине симплекса и выполняет последовательность итераций. В каждой итерации осу-

ществляется переход вдоль ребра симплекса из текущей вершины в соседнюю, целевое значение в которой не меньше (а обычно больше), чем в текущей вершине. Симплекс-алгоритм завершается при достижении локального максимума, т.е. вершины, все соседние вершины которой имеют меньшее целевое значение. Поскольку допустимая область является выпуклой, а целевая функция линейна, локальный оптимум в действительности является глобальным. В разделе 29.4 мы воспользуемся понятием двойственности, чтобы показать, что решение, полученное с помощью симплекс-алгоритма, действительно оптимально.

Хотя геометрическое представление позволяет наглядно проиллюстрировать операции симплекс-алгоритма, мы не будем непосредственно обращаться к нему при подробном рассмотрении симплекс-метода в разделе 29.3. Вместо этого воспользуемся алгебраическим представлением. Сначала запишем задачу линейного программирования в канонической форме в виде набора линейных равенств. Эти линейные равенства выражают одни переменные, называемые *базисными*, через другие переменные, называемые *небазисными*. Переход от одной вершины к другой осуществляется путем замены одной из базисных переменных небазисной переменной. Данная операция называется *замещением* и алгебраически заключается в переписывании задачи линейного программирования в эквивалентной канонической форме.

Приведенный выше пример с двумя переменными был исключительно простым. В данной главе нам предстоит рассмотреть и более сложные случаи: задачи линейного программирования, не имеющие решений; задачи, не имеющие конечного оптимального решения, и задачи, для которых начало координат не является допустимым решением.

Приложения линейного программирования

Линейное программирование имеет широкий спектр приложений. В любом учебнике по исследованию операций содержится множество примеров задач линейного программирования; линейное программирование становится стандартным инструментом, который преподается студентам большинства школ бизнеса. Разработка сценария предвыборной борьбы — лишь один типичный пример. Приведем еще два примера успешного использования линейного программирования.

- Авиакомпания составляет график работы экипажей. Федеральное авиационное агентство установило ряд ограничений, таких как ограничение времени непрерывной работы для каждого члена экипажа и требование, чтобы каждый конкретный экипаж работал только на самолетах одной модели на протяжении одного месяца. Авиакомпания планирует назначить экипажи на все рейсы, задействовав как можно меньше сотрудников (членов экипажей).
- Нефтяная компания выбирает место бурения скважины. С размещением буровой в каждом конкретном месте связаны определенные затраты и ожидаемая прибыль в виде некоторого количества баррелей добытой нефти, рассчитанная на основе проведенных геологических исследований. Средства, выделяемые на бурение новых скважин, ограничены, и компания хочет максимизировать ожидаемое количество добываемой нефти исходя из заданного бюджета.

Задачи линейного программирования полезны также при моделировании и решении задач теории графов и комбинаторных задач, таких как задачи, приведенные в данной книге. Мы уже встречались с частным случаем линейного программирования, который использовался для решения систем разностных ограничений в разделе 24.4. В разделе 29.2 мы покажем, как сформулировать в виде задач линейного программирования некоторые задачи теории графов и задачи транспортных сетей. В разделе 35.4 линейное программирование будет использоваться в качестве инструмента поиска приблизительного решения еще одной задачи теории графов.

Алгоритмы решения задач линейного программирования

В этой главе изучается симплекс-алгоритм. При аккуратном применении данный алгоритм на практике обычно позволяет быстро решать задачи линейного программирования общего вида. Однако при некоторых специально подобранных начальных значениях время выполнения симплекс-алгоритма может оказаться экспоненциальным. Первым алгоритмом с полиномиальным временем выполнения для решения задач линейного программирования был ***эллипсоидный алгоритм***, который на практике работает весьма медленно. Второй класс полиномиальных по времени алгоритмов содержит так называемые ***методы внутренней точки*** (interior-point methods). В отличие от симплекс-алгоритма, который движется по границе допустимой области и на каждой итерации рассматривает допустимое решение, являющееся вершиной симплекса, эти алгоритмы осуществляют движение по внутренней части допустимой области. Промежуточные решения являются допустимыми, но не обязательно находятся в вершинах симплекса, однако конечное решение является вершиной. Для задач большой размерности производительность алгоритмов внутренней точки может быть сравнима с производительностью симплекс-алгоритма, а иногда может и превышать ее. О том, где найти дополнительную информацию по указанным алгоритмам, говорится в заключительных замечаниях к данной главе.

Если ввести в задачу линейного программирования дополнительное требование, чтобы все переменные принимали целые значения, получится задача ***целочисленного линейного программирования***. В упр. 34.5.3 предлагается показать, что в этом случае даже задача поиска допустимого решения является NP-полной; поскольку ни для одной NP-полной задачи полиномиальные алгоритмы решения неизвестны, для задач целочисленного линейного программирования полиномиальные по времени алгоритмы также неизвестны. В отличие от них, задача линейного программирования общего вида может быть решена за полиномиальное время.

В данной главе для указания конкретного набора значений переменных в задаче линейного программирования с переменными $x = (x_1, x_2, \ldots, x_n)$ мы будем использовать обозначение $\bar{x} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$.

29.1. Стандартная и каноническая формы задачи линейного программирования

В данном разделе описываются стандартная и каноническая формы задач линейного программирования, которые будут полезны при формулировании задач и работе с ними. В стандартной форме все ограничения являются неравенствами, а в канонической форме все ограничения являются равенствами (за исключением ограничений, требующих, чтобы все переменные были неотрицательными).

Стандартная форма

В ***стандартной форме*** задаются n действительных чисел $c_1, c_2, \ldots, c_n$; m действительных чисел $b_1, b_2, \ldots, b_m$; и mn действительных чисел a_{ij}, где $i = 1, 2, \ldots, m$ и $j = 1, 2, \ldots, n$. Требуется найти n действительных чисел $x_1, x_2, \ldots, x_n$, которые

максимизируют
$$\sum_{j=1}^{n} c_j x_j \tag{29.16}$$
при условиях
$$\sum_{j=1}^{n} a_{ij} x_j \le b_i \text{ при } i = 1, 2, \ldots, m \tag{29.17}$$
$$x_j \ge 0 \text{ при } j = 1, 2, \ldots, n\,. \tag{29.18}$$

Обобщая введенную для двумерной задачи линейного программирования терминологию, будем называть выражение (29.16) ***целевой функцией***, а $n + m$ неравенств (29.17) и (29.18) — ***ограничениями***; n ограничений (29.18) называются ***ограничениями неотрицательности***. Произвольная задача линейного программирования не обязательно содержит ограничения неотрицательности, но в стандартной форме они необходимы. Иногда удобно записывать задачу линейного программирования в более компактной форме. Определим $m \times n$-матрицу $A = (a_{ij})$, m-мерный вектор $b = (b_i)$, n-мерный вектор $c = (c_j)$ и n-мерный вектор $x = (x_j)$. Тогда задачу линейного программирования (29.16)–(29.18) можно записать в следующем виде:

максимизировать
$$c^{\mathrm{T}} x \tag{29.19}$$
при условиях
$$Ax \le b \tag{29.20}$$
$$x \ge 0\,. \tag{29.21}$$

В строке (29.19) $c^{\mathrm{T}} x$ представляет собой скалярное произведение двух векторов. В неравенстве (29.20) Ax является произведением матрицы и вектора, а $x \ge 0$ в (29.21) означает, что все компоненты вектора x должны быть неотрицательными. Таким образом, задачу линейного программирования в стандартной форме можно описать с помощью тройки (A, b, c), и мы примем соглашение, что A, b, и c всегда имеют указанную выше размерность.

Теперь введем терминологию для описания различных ситуаций, возникающих в линейном программировании. Некоторые термины уже использовались в двумерной задаче. Набор значений переменных $\bar{x}$, которые удовлетворяют всем ограничениям, называется ***допустимым решением***, в то время как набор значений переменных $\bar{x}$, не удовлетворяющий хотя бы одному ограничению, называется ***недопустимым решением***. Решению $\bar{x}$ соответствует ***целевое значение*** $c^{\mathrm{T}}\bar{x}$. Допустимое решение $\bar{x}$, целевое значение которого является максимальным среди всех допустимых решений, является ***оптимальным решением***, а его целевое значение $c^{\mathrm{T}}\bar{x}$ называется ***оптимальным целевым значением***. Если задача линейного программирования не имеет допустимых решений, она называется ***неразрешимой*** (infeasible), в противном случае она является ***разрешимой*** (feasible). Если задача линейного программирования имеет допустимые решения, но не имеет конечного оптимального целевого значения, она называется ***неограниченной*** (unbounded). В упр. 29.1.9 предлагается показать, что задача линейного программирования может иметь конечное оптимальное целевое значение даже в том случае, когда ее допустимая область неограничена.

Преобразование задач линейного программирования в стандартную форму

Любую задачу линейного программирования, в которой требуется минимизировать или максимизировать некую линейную функцию при наличии линейных ограничений, всегда можно преобразовать в стандартную форму. Исходная задача может находиться не в стандартной форме по четырем причинам.

1. Целевая функция минимизируется, а не максимизируется.
2. На некоторые переменные не наложены условия неотрицательности.
3. Некоторые ограничения имеют форму равенств, т.е. имеют знак равенства вместо знака "меньше или равно".
4. Некоторые ограничения-неравенства вместо знака "меньше или равно" имеют знак "больше или равно".

Преобразовывая задачу линейного программирования L в другую задачу линейного программирования L', мы бы хотели, чтобы оптимальное решение задачи L' позволяло найти оптимальное решение задачи L. Будем говорить, что две задачи максимизации, L и L', ***эквивалентны***, если для каждого допустимого решения $\bar{x}$ задачи L с целевым значением z существует соответствующее допустимое решение $\bar{x}'$ задачи L' с целевым значением z, а для каждого допустимого решения $\bar{x}'$ задачи L' с целевым значением z существует соответствующее допустимое решение $\bar{x}$ задачи L с целевым значением z. (Из этого определения не следует взаимно однозначное соответствие между допустимыми решениями.) Задача минимизации L и задача максимизации L' эквивалентны, если для каждого допустимого решения $\bar{x}$ задачи L с целевым значением z существует соответствующее допустимое решение $\bar{x}'$ задачи L' с целевым значением $-z$, а для каждого допустимого решения $\bar{x}'$ задачи L' с целевым значением z существует соответствующее допустимое решение $\bar{x}$ задачи L с целевым значением $-z$.

Теперь покажем, как поочередно избавиться от перечисленных выше возможных проблем. После устранения каждого несоответствия стандартной форме докажем, что новая задача линейного программирования эквивалентна старой.

Чтобы превратить задачу минимизации L в эквивалентную ей задачу максимизации L', достаточно просто изменить знаки коэффициентов целевой функции на противоположные. Поскольку L и L' имеют одинаковые множества допустимых решений и для любого допустимого решения целевое значение L противоположно целевому значению L', эти две задачи линейного программирования эквивалентны. Например, пусть исходная задача имеет вид

$$\begin{aligned} &\text{минимизировать} && -2x_1 + 3x_2 \\ &\text{при условиях} && \\ & && x_1 + x_2 = 7 \\ & && x_1 - 2x_2 \le 4 \\ & && x_1 \ge 0. \end{aligned}$$

Если мы поменяем знаки коэффициентов целевой функции, то получим следующую задачу:

$$\begin{aligned} &\text{максимизировать} && 2x_1 - 3x_2 \\ &\text{при условиях} && \\ & && x_1 + x_2 = 7 \\ & && x_1 - 2x_2 \le 4 \\ & && x_1 \ge 0. \end{aligned}$$

Теперь покажем, как преобразовать задачу линейного программирования, в которой на некоторые переменные не наложены ограничения неотрицательности, в задачу, в которой все переменные подчиняются этому условию. Предположим, что для некоторой переменной x_j ограничение неотрицательности отсутствует. Заменим все вхождения переменной x_j выражением $x'_j - x''_j$ и добавим ограничения неотрицательности $x'_j \ge 0$ и $x''_j \ge 0$. Так, если целевая функция содержит слагаемое $c_j x_j$, то оно заменяется на $c_j x'_j - c_j x''_j$, а если ограничение i содержит слагаемое $a_{ij} x_j$, оно заменяется на $a_{ij} x'_j - a_{ij} x''_j$. Любое допустимое решение $\widehat{x}$ новой задачи линейного программирования соответствует допустимому решению исходной задачи с $\bar{x}_j = \widehat{x}'_j - \widehat{x}''_j$ и тем же самым целевым значением. Точно так же и любое допустимое решение $\bar{x}$ исходной задачи линейного программирования соответствует допустимому решению $\widehat{x}$ новой задачи линейного программирования с $\widehat{x}'_j = \bar{x}_j$ и $\widehat{x}''_j = 0$, если $\bar{x}_j \ge 0$, или с $\widehat{x}''_j = -\bar{x}_j$ и $\widehat{x}'_j = 0$, если $\bar{x}_j < 0$. Две указанные задачи линейного программирования имеют одно и то же целевое значение независимо от знака $\bar{x}_j$. Следовательно, эти две задачи эквивалентны. Применив эту схему преобразования ко всем переменным, для которых нет ограничений неотрицательности, получим эквивалентную задачу линейного программирования, в которой на все переменные наложены ограничения неотрицательности.

Продолжая рассмотрение нашего примера, проверяем, для всех ли переменных есть соответствующие ограничения неотрицательности. Для переменной x_1

такое ограничение есть, а для переменной x_2 — нет. Заменив переменную x_2 переменными x_2' и x_2'' и выполнив соответствующие преобразования, получим следующую задачу:

максимизировать $2x_1 - 3x_2' + 3x_2''$
при условиях

$$\begin{aligned} x_1 + x_2' - x_2'' &= 7 \\ x_1 - 2x_2' + 2x_2'' &\leq 4 \\ x_1, x_2', x_2'' &\geq 0. \end{aligned} \tag{29.22}$$

Теперь преобразуем ограничения-равенства в ограничения-неравенства. Предположим, что задача линейного программирования содержит ограничение-равенство $f(x_1, x_2, \ldots, x_n) = b$. Поскольку $x = y$ тогда и только тогда, когда справедливы оба неравенства, $x \geq y$ и $x \leq y$, можно заменить данное ограничение-равенство парой ограничений-неравенств $f(x_1, x_2, \ldots, x_n) \leq b$ и $f(x_1, x_2, \ldots, x_n) \geq b$. Выполнив такое преобразование для всех ограничений-равенств, получим задачу линейного программирования, в которой все ограничения являются неравенствами.

Наконец можно преобразовать ограничения вида "больше или равно" в ограничения вида "меньше или равно" путем умножения этих ограничений на -1. Любое ограничение вида

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i$$

эквивалентно ограничению

$$\sum_{j=1}^{n} -a_{ij} x_j \leq -b_i \, .$$

Таким образом, заменив каждый коэффициент a_{ij} на $-a_{ij}$ и каждое значение b_j на $-b_j$, мы получим эквивалентное ограничение вида "меньше или равно".

Чтобы завершить преобразование нашего примера, заменим ограничение-равенство (29.22) двумя неравенствами и получим

максимизировать $2x_1 - 3x_2' + 3x_2''$
при условиях

$$\begin{aligned} x_1 + x_2' - x_2'' &\leq 7 \\ x_1 + x_2' - x_2'' &\geq 7 \\ x_1 - 2x_2' + 2x_2'' &\leq 4 \\ x_1, x_2', x_2'' &\geq 0. \end{aligned} \tag{29.23}$$

Теперь изменим знак ограничения (29.23). Для единообразия имен переменных переименуем x_2' в x_2, а x_2'' — в x_3. Полученная стандартная форма имеет вид

максимизировать $2x_1 - 3x_2 + 3x_3$ (29.24)

при условиях

$$x_1 + x_2 - x_3 \leq 7 \quad (29.25)$$

$$-x_1 - x_2 + x_3 \leq -7 \quad (29.26)$$

$$x_1 - 2x_2 + 2x_3 \leq 4 \quad (29.27)$$

$$x_1, x_2, x_3 \geq 0. \quad (29.28)$$

Преобразование задач линейного программирования в каноническую форму

Чтобы эффективно решать задачу линейного программирования с помощью симплекс-метода, удобно записать ее в такой форме, когда некоторые ограничения заданы в виде равенств. Говоря более точно, мы будем приводить задачу к форме, в которой только ограничения неотрицательности заданы в виде неравенств, а остальные ограничения являются равенствами. Пусть ограничение-неравенство имеет вид

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i \ . \quad (29.29)$$

Введем новую переменную s и перепишем неравенство (29.29) в виде двух ограничений:

$$s = b_i - \sum_{j=1}^{n} a_{ij}x_j \ , \quad (29.30)$$

$$s \geq 0 \ . \quad (29.31)$$

Переменная s называется ***вспомогательной переменной*** (slack variable), так как она определяет ***разность*** (slack) между левой и правой частями выражения (29.29). (Вскоре вы узнаете, почему удобно записывать ограничения в виде, когда в левой части находятся только вспомогательные переменные.) Поскольку неравенство (29.29) верно тогда и только тогда, когда одновременно выполнены равенство (29.30) и неравенство (29.31), можно применить данное преобразование ко всем ограничениям-неравенствам задачи линейного программирования и получить эквивалентную задачу, в которой в виде неравенств записаны только условия неотрицательности. При переходе от стандартной формы к канонической мы будем использовать для связанной с i-м ограничением вспомогательной переменной обозначение x_{n+i} (вместо s). Тогда i-е ограничение будет записано в виде равенства

$$x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij}x_j \quad (29.32)$$

наряду с ограничением неотрицательности $x_{n+i} \geq 0$.

Применив данное преобразование ко всем ограничениям задачи линейного программирования в стандартной форме, получаем задачу в канонической форме. Например, для задачи, заданной формулами (29.24)–(29.28), введя вспомогательные переменные x_4, x_5 и x_6, получим

максимизировать
$$2x_1 - 3x_2 + 3x_3 \tag{29.33}$$
при условиях
$$x_4 = 7 - x_1 - x_2 + x_3 \tag{29.34}$$
$$x_5 = -7 + x_1 + x_2 - x_3 \tag{29.35}$$
$$x_6 = 4 - x_1 + 2x_2 - 2x_3 \tag{29.36}$$
$$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 . \tag{29.37}$$

В этой задаче линейного программирования все ограничения, за исключением условий неотрицательности, являются равенствами и все переменные подчиняются ограничениям неотрицательности. В записи каждого ограничения-равенства в левой части находится одна переменная, а остальные переменные находятся в правой части. Более того, в правой части каждого уравнения содержатся одни и те же переменные, и это только те переменные, которые входят в целевую функцию. Переменные, находящиеся в левой части равенств, называются ***базисными переменными*** (basic variables), а переменные, находящиеся в правой части, – ***небазисными переменными*** (nonbasic variables).

В задачах линейного программирования, удовлетворяющих указанным условиям, мы будем иногда опускать слова "максимизировать" и "при условиях", а также явные ограничения неотрицательности. Для обозначения значения целевой функции мы будем использовать переменную z. Полученная форма записи называется ***канонической формой*** (slack form). Если записать задачу линейного программирования (29.33)–(29.37) в канонической форме, можно получить

$$z = 2x_1 - 3x_2 + 3x_3 \tag{29.38}$$
$$x_4 = 7 - x_1 - x_2 + x_3 \tag{29.39}$$
$$x_5 = -7 + x_1 + x_2 - x_3 \tag{29.40}$$
$$x_6 = 4 - x_1 + 2x_2 - 2x_3 . \tag{29.41}$$

Для канонической формы, как и для стандартной, удобно иметь более короткий способ записи. Как будет показано в разделе 29.3, множества базисных и небазисных переменных в процессе работы симплекс-алгоритма будут меняться. Обозначим множество индексов небазисных переменных как N, а множество индексов базисных переменных – как B. Всегда выполняются соотношения $|N| = n$, $|B| = m$ и $N \cup B = \{1, 2, \ldots, n + m\}$. Уравнения будут индексироваться элементами множества B, а переменные правых частей будут индексироваться элементами множества N. Как и в стандартной форме, мы используем b_j, c_j и a_{ij} для обозначения констант и коэффициентов. Для обозначения необязательного постоянного члена в целевой функции используется v (позже вы узнаете, что включение константного члена в целевую функцию упрощает определение ее значения). Таким образом, можно кратко описать каноническую форму с помо-

щью кортежа (N, B, A, b, c, v); она служит кратким обозначением канонической формы

$$z = v + \sum_{j \in N} c_j x_j \tag{29.42}$$

$$x_i = b_i - \sum_{j \in N} a_{ij} x_j \quad \text{для } i \in B\,, \tag{29.43}$$

в которой все переменные x подчиняются условиям неотрицательности. Поскольку сумма $\sum_{j \in N} a_{ij} x_j$ в выражении (29.43) вычитается, значения элементов a_{ij} противоположны коэффициентам, входящим в каноническую форму.

Например, для канонической формы

$$\begin{aligned} z &= 28 - \frac{x_3}{6} - \frac{x_5}{6} - \frac{2x_6}{3} \\ x_1 &= 8 + \frac{x_3}{6} + \frac{x_5}{6} - \frac{x_6}{3} \\ x_2 &= 4 - \frac{8x_3}{3} - \frac{2x_5}{3} + \frac{x_6}{3} \\ x_4 &= 18 - \frac{x_3}{2} + \frac{x_5}{2} \end{aligned}$$

мы имеем $B = \{1, 2, 4\}$, $N = \{3, 5, 6\}$,

$$A = \begin{pmatrix} a_{13} & a_{15} & a_{16} \\ a_{23} & a_{25} & a_{26} \\ a_{43} & a_{45} & a_{46} \end{pmatrix} = \begin{pmatrix} -1/6 & -1/6 & 1/3 \\ 8/3 & 2/3 & -1/3 \\ 1/2 & -1/2 & 0 \end{pmatrix},$$

$$b = \begin{pmatrix} b_1 \\ b_2 \\ b_4 \end{pmatrix} = \begin{pmatrix} 8 \\ 4 \\ 18 \end{pmatrix},$$

$c = \begin{pmatrix} c_3 & c_5 & c_6 \end{pmatrix}^{\mathrm{T}} = \begin{pmatrix} -1/6 & -1/6 & -2/3 \end{pmatrix}^{\mathrm{T}}$ и $v = 28$. Заметим, что индексы у A, b и c необязательно являются множествами последовательных целых чисел; они зависят от того, какие индексы входят в множества B и N. В качестве примера противоположности элементов матрицы A коэффициентам канонической формы заметим, что в уравнение для x_1 входит слагаемое $x_3/6$, так что коэффициент a_{13} равен $-1/6$, а не $+1/6$.

Упражнения

29.1.1

Если записать задачу линейного программирования (29.24)–(29.28) в компактном виде (29.19)–(29.21), то чему будут равны n, m, A, b и c?

29.1.2

Укажите три допустимых решения задачи линейного программирования (29.24)–(29.28). Чему равно целевое значение для каждого решения?

29.1.3

Каковы N, B, A, b, c и v для канонической формы (29.38)–(29.41)?

29.1.4

Приведите следующую задачу линейного программирования к стандартной форме:

минимизировать $2x_1 + 7x_2 + x_3$
при условиях
$$\begin{aligned} x_1 - x_3 &= 7 \\ 3x_1 + x_2 &\geq 24 \\ x_2 &\geq 0 \\ x_3 &\leq 0. \end{aligned}$$

29.1.5

Приведите следующую задачу линейного программирования к канонической форме:

максимизировать $2x_1 - 6x_3$
при условиях
$$\begin{aligned} x_1 + x_2 - x_3 &\leq 7 \\ 3x_1 - x_2 &\geq 8 \\ -x_1 + 2x_2 + 2x_3 &\geq 0 \\ x_1, x_2, x_3 &\geq 0. \end{aligned}$$

Какие переменные являются базисными, а какие небазисными?

29.1.6

Покажите, что следующая задача линейного программирования является неразрешимой:

максимизировать $3x_1 - 2x_2$
при условиях
$$\begin{aligned} x_1 + x_2 &\leq 2 \\ -2x_1 - 2x_2 &\leq -10 \\ x_1, x_2 &\geq 0. \end{aligned}$$

29.1.7

Покажите, что следующая задача линейного программирования является неограниченной:

максимизировать $x_1 - x_2$

при условиях

$$\begin{aligned} -2x_1 + x_2 &\leq -1 \\ -x_1 - 2x_2 &\leq -2 \\ x_1, x_2 &\geq 0. \end{aligned}$$

29.1.8

Пусть имеется задача линейного программирования общего вида с n переменными и m ограничениями. Предположим, что мы преобразовали ее в стандартную форму. Укажите верхнюю границу числа переменных и ограничений в полученной задаче.

29.1.9

Приведите пример задачи линейного программирования, для которой допустимая область является неограниченной, но оптимальное целевое значение конечно.

29.2. Формулировка задач в виде задач линейного программирования

Хотя основное внимание в данной главе уделяется симплекс-алгоритму, важно также понимать, в каких случаях задачу можно сформулировать в виде задачи линейного программирования. После того как задача сформулирована в этом виде, ее можно решить за полиномиальное время с помощью эллипсоидного алгоритма, или алгоритма внутренней точки. Существует несколько пакетов прикладных программ, эффективно решающих задачи линейного программирования, а значит, если проблему удастся сформулировать в виде задачи линейного программирования, то ее можно решить на практике с помощью такого пакета.

Рассмотрим несколько конкретных примеров задач линейного программирования. Начнем с двух уже рассмотренных ранее задач: задачи поиска кратчайших путей из одной вершины (см. главу 24) и задачи поиска максимального потока (см. главу 26). После этого мы опишем задачу поиска потока с минимальной стоимостью. Для данной задачи существует алгоритм с полиномиальными затратами времени, не основанный на линейном программировании, однако мы не будем его рассматривать. И наконец мы опишем задачу многопродуктного потока (multicomodity-flow problem), единственный известный полиномиальный по времени алгоритм решения которой базируется на линейном программировании.

При решении задач с графами в части VI мы использовали запись атрибутов в виде $v.d$ и $(u, v).f$. Однако линейное программирование обычно использует не присоединенные атрибуты, а переменные с нижними индексами. Таким образом, выражая переменные в задачах линейного программирования, необходимо указы-

вать вершины и ребра с помощью индексов. Например, вес кратчайшего пути для вершины v обозначается не как $v.d$, а как d_v. Аналогично поток из вершины u в вершину v обозначается не как $(u,v).f$, а как f_{uv}. Для величин, являющихся входными данными для задач, таких как веса или пропускные способности ребер, мы будем продолжать использовать записи $w(u,v)$ и $c(u,v)$.

Кратчайшие пути

Задачу поиска кратчайших путей из одной вершины-источника можно сформулировать в виде задачи линейного программирования. В данном разделе мы остановимся на формулировке задачи кратчайшего пути для одной пары, а более общий случай задачи поиска кратчайших путей из одного источника предлагается рассмотреть в качестве упр. 29.2.3.

В задаче поиска кратчайшего пути для одной пары у нас имеются взвешенный ориентированный граф $G = (V,E)$, весовая функция $w : E \to \mathbb{R}$, ставящая в соответствие ребрам графа действительные веса, исходная вершина s и вершина назначения t. Мы хотим вычислить значение d_t, которое является весом кратчайшего пути из s в t. Чтобы записать эту задачу в виде задачи линейного программирования, необходимо определить множество переменных и ограничения, которые позволят определить, какой путь из s в t является кратчайшим. К счастью, алгоритм Беллмана–Форда именно это и делает. Когда алгоритм Беллмана–Форда завершает свою работу, для каждой вершины v оказывается вычисленным значение d_v (мы используем запись с индексом, а не атрибут), такое, что для каждого ребра $(u,v) \in E$ выполняется условие $d_v \le d_u + w(u,v)$. Исходная вершина изначально получает значение $d_s = 0$, которое никогда не изменяется. Таким образом, мы получаем следующую задачу линейного программирования для вычисления веса кратчайшего пути из s в t:

максимизировать d_t (29.44)

при условиях

$$d_v \le d_u + w(u,v) \text{ для каждого ребра } (u,v) \in E\,, \tag{29.45}$$

$$d_s = 0\,. \tag{29.46}$$

Вас может удивить то, что данная задача линейного программирования максимизирует целевую функцию, в то время как предполагается, что задача вычисляет кратчайшие пути. Мы не хотим минимизировать целевую функцию, поскольку тогда присваивание $\bar{d}_v = 0$ для всех $v \in V$ даст оптимальное решение задачи линейного программирования без решения задачи поиска кратчайшего пути. Мы выполняем максимизацию, поскольку оптимальное решение задачи поиска кратчайших путей присваивает каждому $\bar{d}_v$ значение $\min_{u:(u,v)\in E}\{\bar{d}_u + w(u,v)\}$, так что $\bar{d}_v$ является наибольшим значением, которое не превышает все значения множества $\{\bar{d}_u + w(u,v)\}$. Мы хотим максимизировать d_v для всех вершин v на кратчайшем пути из s в t при условии выполнения указанных ограничений для всех вершин v, и максимизация d_t достигает этой цели.

Эта задача линейного программирования имеет $|V|$ переменных d_v, по одной для каждой вершины $v \in V$. В ней также имеется $|E|+1$ ограничений: по одному

для каждого ребра плюс дополнительное ограничение, что вес исходной вершины кратчайшего пути всегда имеет значение 0.

Максимальный поток

Задачу поиска максимального потока также можно сформулировать в виде задачи линейного программирования. Напомним, что в ней задаются ориентированный граф $G = (V, E)$, в котором каждое ребро $(u, v) \in E$ имеет неотрицательную пропускную способность $c(u, v) \geq 0$, и две различные вершины, источник s и сток t. Согласно определению из раздела 26.1 поток является действительной функцией $f : V \times V \to \mathbb{R}$, удовлетворяющей ограничениям пропускной способности и сохранения потока. Максимальный поток представляет собой поток, который удовлетворяет данным ограничениям и максимизирует величину потока, представляющую собой суммарный поток, выходящий из источника, минус суммарный поток, входящий в источник. Таким образом, поток удовлетворяет линейным ограничениям, а величина потока является линейной функцией. Напомним также, что мы предполагаем, что $c(u, v) = 0$, если $(u, v) \notin E$. Теперь можно записать задачу максимизации потока в виде задачи линейного программирования:

$$\begin{aligned}
&\text{максимизировать} && \sum_{v \in V} f_{sv} - \sum_{v \in V} f_{vs} && && (29.47)\\
&\text{при условиях} && f_{uv} \leq c(u, v) && \text{для всех } u, v \in V\ , && (29.48)\\
& && \sum_{v \in V} f_{vu} = \sum_{v \in V} f_{uv} && \text{для каждой вершины } u \in V - \{s, t\}\ , && (29.49)\\
& && f_{uv} \geq 0 && \text{для всех } u, v \in V\ . && (29.50)
\end{aligned}$$

Эта задача линейного программирования имеет $|V|^2$ переменных, соответствующих потоку между каждой парой вершин, и $2|V|^2 + |V| - 2$ ограничений.

Обычно более эффективно решаются задачи линейного программирования меньшей размерности. В задаче (29.47)–(29.50) для простоты записи считается, что поток и пропускная способность каждой пары вершин u, v, таких, что $(u, v) \notin E$, равны 0. Но было бы эффективнее переписать задачу так, чтобы в ней содержалось $O(V + E)$ ограничений, что и предлагается сделать в упр. 29.2.5.

Поиск потока минимальной стоимости

До сих пор в данном разделе использовалось линейное программирование для решения задач, для которых уже известны эффективные алгоритмы. Фактически хороший алгоритм, разработанный специально для конкретной задачи, например алгоритм Дейкстры для задачи поиска кратчайшего пути из одной вершины или метод проталкивания для задачи максимального потока, обычно более эффективен, чем линейное программирование, — как в теории, так и на практике.

Истинная ценность линейного программирования состоит в возможности решения новых задач. Вспомним задачу подготовки к предвыборной кампании, описанную в начале данной главы. Задача получения необходимого количества

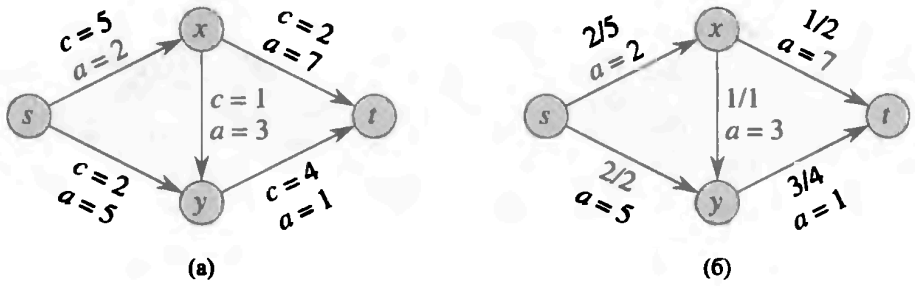

Рис. 29.3. **(а)** Пример задачи поиска потока минимальной стоимости. Пропускные способности обозначены как c, а стоимости — как a. Вершина s является источником, а вершина t — стоком, и необходимо переслать четыре единицы потока из s в t. **(б)** Решение задачи поиска потока минимальной стоимости, в которой из истока s в сток t пересылаются четыре единицы потока. Для каждого ребра поток и пропускная способность указаны в виде "поток/пропускная способность".

голосов при минимальных затратах не решается ни одним из алгоритмов, уже изученных нами в данной книге, однако ее можно решить с помощью линейного программирования. О реальных задачах, которые можно решить с помощью линейного программирования, написано множество книг. Линейное программирование также чрезвычайно полезно при решении разновидностей задач, для которых еще не известны эффективные алгоритмы.

Рассмотрим, например, следующее обобщение задачи поиска максимального потока. Предположим, что каждому ребру (u, v), помимо пропускной способности $c(u, v)$, соответствует некая стоимость $a(u, v)$, принимающая действительные значения. Как и в задаче поиска максимального потока, мы считаем, что $c(u, v) = 0$, если $(u, v) \notin E$, и что антипараллельных ребер в графе нет. Если по ребру (u, v) послано f_{uv} единиц потока, это приводит к стоимости пересылки $a(u, v)f_{uv}$. Также задано целевое значение потока d. Необходимо переправить d единиц потока из s в t таким образом, чтобы общая стоимость, связанная с передачей потока, $\sum_{(u,v)\in E} a(u, v)f_{uv}$, была минимальной. Эта задача называется ***задачей поиска потока минимальной стоимости***.

На рис. 29.3, (а) представлен пример задачи поиска потока минимальной стоимости. Нужно переслать четыре единицы потока из s в t, минимизировав суммарную стоимость (пропускная способность каждого ребра обозначена как c, а стоимость — как a). С любым допустимым потоком, т.е. с функцией f, удовлетворяющей ограничениям (29.48)–(29.50), связана стоимость $\sum_{(u,v)\in E} a(u, v)f_{uv}$. Необходимо найти такой поток, который минимизирует эту стоимость. Оптимальное решение показано на рис. 29.3, (б); ему соответствует суммарная стоимость $\sum_{(u,v)\in E} a(u, v)f_{uv} = (2 \cdot 2) + (5 \cdot 2) + (3 \cdot 1) + (7 \cdot 1) + (1 \cdot 3) = 27$.

Существуют алгоритмы с полиномиальными затратами времени, специально разработанные для задачи поиска потока минимальной стоимости, но они выходят за рамки данной книги. Запишем рассматриваемую задачу в виде задачи линейного программирования. Эта задача линейного программирования напоминает задачу, записанную для максимизации потока, однако содержит дополнительное ограничение, состоящее в том, что величина потока составляет ровно d единиц,

и новую целевую функцию минимизации стоимости:

$$\begin{aligned}
&\text{минимизировать} && \sum_{(u,v)\in E} a(u,v)f_{uv} && && (29.51)\\
&\text{при условиях} && f_{uv} \le c(u,v) && \text{для всех } u,v \in V\ , \\
& && \sum_{v\in V} f_{vu} - \sum_{v\in V} f_{uv} = 0 && \text{для каждой вершины } u \in V - \{s,t\}\ , \\
& && \sum_{v\in V} f_{sv} - \sum_{v\in V} f_{vs} = d\ , \\
& && f_{uv} \ge 0 && \text{для всех } u,v \in V\ . && (29.52)
\end{aligned}$$

Многопродуктный поток

В качестве последнего примера рассмотрим еще одну задачу поиска потоков. Предположим, что компания Lucky Puck из раздела 26.1 приняла решение о расширении ассортимента продукции и отгружает не только хоккейные шайбы, но также клюшки и шлемы. Каждый элемент снаряжения выпускается на отдельной фабрике, хранится на отдельном складе и должен ежедневно доставляться с фабрики на склад. Клюшки производятся в Ванкувере и доставляются в Саскатун, а шлемы производятся в Эдмонтоне и доставляются в Реджину. Однако пропускная способность сети доставки не изменилась, и различные элементы, или ***продукты***, должны совместно использовать одну и ту же сеть.

Данный пример представляет собой экземпляр задачи ***многопродуктного потока*** (multicommodity flow). В этой задаче снова задан ориентированный граф $G = (V, E)$, в котором каждое ребро $(u, v) \in E$ имеет неотрицательную пропускную способность $c(u, v) \ge 0$. Как и в задаче поиска максимального потока, неявно подразумевается, что $c(u, v) = 0$ для $(u, v) \notin E$ и что в графе нет антипараллельных ребер. Кроме того, даны k различных продуктов $K_1, K_2, \ldots, K_k$, причем каждый i-й продукт характеризуется тройкой $K_i = (s_i, t_i, d_i)$, где s_i — источник продукта i, t_i — место назначения продукта i, а d_i — спрос, т.е. желаемая величина потока продукта i из s_i в t_i. По определению поток продукта i, обозначаемый f_i (так что f_{iuv} представляет собой поток продукта i из вершины u в вершину v), — это действительная функция, удовлетворяющая ограничениям сохранения потока и пропускной способности. Определим ***совокупный поток*** (aggregate flow) f_{uv} как сумму потоков различных продуктов: $f_{uv} = \sum_{i=1}^{k} f_{iuv}$. Совокупный поток по ребру (u, v) не должен превышать пропускную способность ребра (u, v). При таком способе описания задачи не нужно ничего минимизировать; необходимо только определить, можно ли найти такой поток. Поэтому мы записываем

задачу линейного программирования с "пустой" целевой функцией:

минимизировать 0

при условиях

$$\sum_{i=1}^{k} f_{iuv} \le c(u,v) \quad \text{для всех } u,v \in V ,$$

$$\sum_{v \in V} f_{iuv} - \sum_{v \in V} f_{ivu} = 0 \quad \text{для всех } i = 1,2,\ldots,k \text{ и для всех } u \in V - \{s_i, t_i\} ,$$

$$\sum_{v \in V} f_{i,s_i,v} - \sum_{v \in V} f_{i,v,s_i} = d_i \quad \text{для всех } i = 1,2,\ldots,k ,$$

$$f_{iuv} \ge 0 \quad \text{для всех } u,v \in V \text{ и для всех } i = 1,2,\ldots,k .$$

Единственный известный алгоритм решения этой задачи с полиномиальным временем выполнения состоит в том, чтобы записать ее в виде задачи линейного программирования, а затем решить с помощью полиномиального по времени алгоритма линейного программирования.

Упражнения

29.2.1

Приведите задачу линейного программирования поиска кратчайшего пути для одной пары вершин, заданную формулами (29.44)–(29.46), к стандартной форме.

29.2.2

Запишите в явном виде задачу линейного программирования, соответствующую задаче поиска кратчайшего пути из узла s в узел y на рис. 24.2, (а).

29.2.3

В задаче поиска кратчайшего пути из одной вершины необходимо найти веса кратчайших путей из вершины s во все вершины $v \in V$. Запишите задачу линейного программирования для данного графа G, решение которой обладает тем свойством, что d_v является весом кратчайшего пути из s в v для всех вершин $v \in V$.

29.2.4

Запишите задачу линейного программирования, соответствующую поиску максимального потока на рис. 26.1, (а).

29.2.5

Перепишите задачу линейного программирования для поиска максимального потока (29.47)–(29.50) так, чтобы в ней было только $O(V + E)$ ограничений.

29.2.6

Сформулируйте задачу линейного программирования, которая для заданного двудольного графа $G = (V, E)$ решает задачу о взвешенных паросочетаниях с максимальным весом (maximum-bipartite-matching).

29.2.7

В задаче поиска ***многопродуктного потока с минимальной стоимостью*** (minimum-cost multicommodity-flow problem) задан ориентированный граф $G = (V, E)$, в котором каждому ребру $(u, v) \in E$ соответствуют неотрицательная пропускная способность $c(u, v) \geq 0$ и стоимость $a(u, v)$. Как и в задаче многопродуктного потока, имеется k различных товаров $K_1, K_2, \ldots, K_k$, при этом каждый продукт i характеризуется тройкой $K_i = (s_i, t_i, d_i)$. Поток f_i для i-го продукта и совокупный поток f_{uv} вдоль ребра (u, v) определяются так же, как и в задаче многопродуктного потока. Допустимым является такой поток, для которого совокупный поток вдоль каждого ребра (u, v) не превышает его пропускной способности. Стоимость потока определяется как $\sum_{u,v \in V} a(u, v) f_{uv}$, и цель состоит в том, чтобы найти допустимый поток с минимальной стоимостью. Запишите данную задачу в виде задачи линейного программирования.

29.3. Симплекс-алгоритм

Симплекс-алгоритм является классическим методом решения задач линейного программирования. В отличие от многих других приведенных в данной книге алгоритмов, время выполнения этого алгоритма в худшем случае не является полиномиальным. Однако он действительно выражает суть линейного программирования и на практике зачастую бывает замечательно быстрым.

В дополнение к описанной ранее геометрической интерпретации можно провести определенные аналогии между ним и методом исключения Гаусса, рассмотренным в разделе 28.1. Метод исключения Гаусса применяется при поиске решения системы линейных уравнений. На каждой итерации система переписывается в эквивалентной форме, имеющей определенную структуру. После ряда итераций получается система, записанная в таком виде, что можно легко найти ее решение. Симплекс-алгоритм работает аналогичным образом, и его можно рассматривать как метод исключения Гаусса для неравенств.

Опишем основную идею, лежащую в основе итераций симплекс-алгоритма. С каждой итерацией связывается некое "базисное решение", которое легко получить из канонической формы задачи линейного программирования: каждой небазисной переменной присваивается значение 0, и из ограничений-равенств вычисляются значения базисных переменных. Базисное решение всегда соответствует некой вершине симплекса. С алгебраической точки зрения каждая итерация преобразует одну каноническую форму в эквивалентную. Целевое значение, соответствующее новому базисному допустимому решению, должно быть не меньше (как правило, больше) целевого значения предыдущей итерации. Чтобы добиться

этого увеличения, выбирается некоторая небазисная переменная, причем такая, что при увеличении ее значения от нуля целевое значение также увеличится. То, насколько можно увеличить данную переменную, определяется другими ограничениями, а именно — ее значение увеличивается до тех пор, пока какая-либо базисная переменная не станет равной нулю. После этого каноническая форма переписывается так, что эта базисная переменная и выбранная небазисная переменная меняются ролями. Хотя мы использовали определенные начальные значения переменных для описания алгоритма и будем использовать их в наших доказательствах, в алгоритме данное решение в явном виде не поддерживается. Он просто переписывает задачу линейного программирования до тех пор, пока оптимальное решение не станет "очевидным".

Пример симплекс-алгоритма

Начнем с конкретного примера. Рассмотрим следующую задачу линейного программирования в стандартной форме:

максимизировать
$$3x_1 + x_2 + 2x_3 \tag{29.53}$$
при условиях
$$x_1 + x_2 + 3x_3 \leq 30 \tag{29.54}$$
$$2x_1 + 2x_2 + 5x_3 \leq 24 \tag{29.55}$$
$$4x_1 + x_2 + 2x_3 \leq 36 \tag{29.56}$$
$$x_1, x_2, x_3 \geq 0. \tag{29.57}$$

Чтобы можно было применить симплекс-алгоритм, необходимо преобразовать данную задачу в каноническую форму, как описано в разделе 29.1. Вспомогательные переменные — это не просто элементы алгебраического преобразования, они являются содержательным алгоритмическим понятием. Напомним (см. раздел 29.1), что каждой переменной соответствует ограничение неотрицательности. Будем говорить, что ограничение-равенство является ***строгим*** (tight) при определенном наборе значений его небазисных переменных, если при этих значениях базисная переменная данного ограничения становится равной нулю. Аналогично набор значений небазисных переменных, который делает базисную переменную отрицательной, ***нарушает*** данное ограничение. Таким образом, вспомогательные переменные наглядно показывают, насколько сильно каждое ограничение отличается от строгого, и тем самым помогают определить, насколько можно увеличить значения небазисных переменных, не нарушив ни одного ограничения.

Свяжем с ограничениями (29.54)–(29.56) вспомогательные переменные x_4, x_5 и x_6 соответственно и приведем задачу линейного программирования к канонической форме. В результате получится следующая задача:

$$z = 3x_1 + x_2 + 2x_3 \tag{29.58}$$
$$x_4 = 30 - x_1 - x_2 - 3x_3 \tag{29.59}$$
$$x_5 = 24 - 2x_1 - 2x_2 - 5x_3 \tag{29.60}$$
$$x_6 = 36 - 4x_1 - x_2 - 2x_3 . \tag{29.61}$$

Система ограничений (29.59)–(29.61) содержит три уравнения и шесть переменных. Любое задание значений переменных x_1, x_2 и x_3 определяет значения переменных x_4, x_5 и x_6, следовательно, существует бесконечное число решений данной системы уравнений. Решение является допустимым, если все $x_1, x_2, \ldots, x_6$ неотрицательны. Число допустимых решений также может быть бесконечным. Свойство бесконечности числа возможных решений подобной системы понадобится нам в дальнейших доказательствах. Рассмотрим ***базисное решение*** (basic solution): установим все (небазисные) переменные правой части равными нулю и вычислим значения (базисных) переменных левой части. В данном примере базисным решением является $(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_6) = (0, 0, 0, 30, 24, 36)$, и ему соответствует целевое значение $z = (3 \cdot 0) + (1 \cdot 0) + (2 \cdot 0) = 0$. Заметим, что в этом базисном решении $\bar{x}_i = b_i$ для всех $i \in B$. Итерация симплекс-алгоритма переписывает множество уравнений и целевую функцию так, что в правой части оказывается другое множество переменных. Таким образом, с переписанной задачей связано другое базисное решение. Мы подчеркиваем, что такая перезапись никоим образом не меняет лежащую в основе задачу линейного программирования; задача на каждой итерации имеет точно то же множество допустимых решений, что и задача на предыдущей итерации. Однако эта задача имеет базисное решение, отличное от базисного решения предыдущей итерации.

Если базисное решение является также допустимым, оно называется ***допустимым базисным решением***. В процессе работы симплекс-алгоритма базисное решение практически всегда будет допустимым базисным решением. Однако в разделе 29.5 мы покажем, что в нескольких первых итерациях симплекс-алгоритма базисное решение может не быть допустимым.

На каждой итерации нашей целью является переформулировка задачи линейного программирования таким образом, чтобы новое базисное решение имело большее целевое значение. Мы выбираем некоторую небазисную переменную x_e, коэффициент при которой в целевой функции положителен, и увеличиваем ее значение настолько, насколько это возможно без нарушения существующих ограничений. Переменная x_e становится базисной, а некоторая другая переменная x_l становится небазисной. Значения остальных базисных переменных и целевой функции также могут измениться.

Продолжая изучение примера, рассмотрим возможность увеличения значения x_1. При увеличении x_1 значения переменных x_4, x_5 и x_6 уменьшаются. Поскольку на каждую переменную наложено ограничение неотрицательности, ни одна из этих переменных не должна стать отрицательной. Если x_1 увеличить более чем на 30, то x_4 станет отрицательным, а x_5 и x_6 станут отрицательными при увеличении x_1 на 12 и 9 соответственно. Третье ограничение (29.61) является самым строгим, именно оно определяет, насколько можно увеличить x_1. Следовательно, меняем ролями переменные x_1 и x_6. Решив уравнение (29.61) относительно x_1, получим

$$x_1 = 9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4} \,. \tag{29.62}$$

Чтобы записать другие уравнения с x_6 в правой части, подставим вместо x_1 выражение из (29.62). Для уравнения (29.59) получаем

$$\begin{aligned} x_4 &= 30 - x_1 - x_2 - 3x_3 \\ &= 30 - \left(9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4}\right) - x_2 - 3x_3 \\ &= 21 - \frac{3x_2}{4} - \frac{5x_3}{2} + \frac{x_6}{4} . \end{aligned} \tag{29.63}$$

Аналогично комбинируем уравнение (29.62) с ограничением (29.60) и целевой функцией (29.58) и записываем нашу задачу линейного программирования в следующем виде:

$$z = 27 + \frac{x_2}{4} + \frac{x_3}{2} - \frac{3x_6}{4} \tag{29.64}$$

$$x_1 = 9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4} \tag{29.65}$$

$$x_4 = 21 - \frac{3x_2}{4} - \frac{5x_3}{2} + \frac{x_6}{4} \tag{29.66}$$

$$x_5 = 6 - \frac{3x_2}{2} - 4x_3 + \frac{x_6}{2} . \tag{29.67}$$

Эта операция называется ***замещением***. Как было показано выше, в процессе замещения выбираются небазисная переменная x_e, называемая ***вводимой переменной*** (entering variable), и базисная переменная x_l, называемая ***выводимой переменной*** (leaving variable), которые затем меняются ролями.

Задача, записанная уравнениями (29.64)–(29.67), эквивалентна задаче 29.58–29.61. В процессе работы симплекс-алгоритма выполняются только операции переноса переменных из левой части уравнения в правую и обратно, а также подстановки одного уравнения в другое. Первая операция очевидным образом создает эквивалентную задачу, то же самое можно сказать и о второй операции (см. упр. 29.3.3).

Чтобы продемонстрировать эквивалентность указанных задач, убедимся, что исходное базисное решение $(0, 0, 0, 30, 24, 36)$ удовлетворяет новым уравнениям (29.65)–(29.67) и имеет целевое значение $27+(1/4)\cdot 0+(1/2)\cdot 0-(3/4)\cdot 36 = 0$. В базисном решении, связанном с новой задачей, новые небазисные переменные равны нулю. Таким образом, оно имеет вид $(9, 0, 0, 21, 6, 0)$, а соответствующее целевое значение $z = 27$. Простые арифметические действия позволяют убедиться, что данное решение также удовлетворяет уравнениям (29.59)–(29.61) и при подстановке в целевую функцию (29.58) имеет целевое значение $(3 \cdot 9) + (1 \cdot 0) + (2 \cdot 0) = 27$.

Продолжая рассмотрение примера, необходимо найти новую базисную переменную, значение которой можно увеличить. Нет смысла увеличивать x_6, поскольку при ее увеличении целевое значение уменьшается. Можно попробовать увеличить x_2 или x_3; мы выберем x_3. Насколько можно увеличить x_3, чтобы не нарушить ни одно из ограничений? Ограничение (29.65) допускает увеличение, не превышающее 18, ограничение (29.66) — $42/5$, а ограничение (29.67) — $3/2$.

Третье ограничение снова оказывается самым строгим, следовательно, мы переписываем его так, чтобы x_3 было в левой части, а x_5 — в правой. Затем подставляем это новое уравнение $x_3 = 3/2 - 3x_2/8 - x_5/4 + x_6/8$ в уравнения (29.64)–(29.66) и получаем новую эквивалентную задачу:

$$z = \frac{111}{4} + \frac{x_2}{16} - \frac{x_5}{8} - \frac{11x_6}{16} \tag{29.68}$$

$$x_1 = \frac{33}{4} - \frac{x_2}{16} + \frac{x_5}{8} - \frac{5x_6}{16} \tag{29.69}$$

$$x_3 = \frac{3}{2} - \frac{3x_2}{8} - \frac{x_5}{4} + \frac{x_6}{8} \tag{29.70}$$

$$x_4 = \frac{69}{4} + \frac{3x_2}{16} + \frac{5x_5}{8} - \frac{x_6}{16}\,. \tag{29.71}$$

С этой системой связано базисное решение $(33/4, 0, 3/2, 69/4, 0, 0)$ с целевым значением $111/4$. Теперь единственная возможность увеличить целевое значение — увеличить x_2. Имеющиеся ограничения задают верхние границы увеличения 132, 4 и ∞ соответственно. (Верхняя граница в ограничении (29.71) равна ∞, поскольку при увеличении x_2 значение базисной переменной x_4 также увеличивается. Следовательно, данное уравнение не налагает никаких ограничений на величину возможного увеличения x_2.) Увеличиваем x_2 до 4 и делаем эту переменную базисной. Затем решаем уравнение (29.70) относительно x_2, подставляем полученное выражения в другие уравнения и получаем новую задачу:

$$z = 28 - \frac{x_3}{6} - \frac{x_5}{6} - \frac{2x_6}{3} \tag{29.72}$$

$$x_1 = 8 + \frac{x_3}{6} + \frac{x_5}{6} - \frac{x_6}{3} \tag{29.73}$$

$$x_2 = 4 - \frac{8x_3}{3} - \frac{2x_5}{3} + \frac{x_6}{3} \tag{29.74}$$

$$x_4 = 18 - \frac{x_3}{2} + \frac{x_5}{2}\,. \tag{29.75}$$

В полученной задаче все коэффициенты целевой функции отрицательны. Как будет показано далее, такая ситуация возникает только тогда, когда базисное решение переписанной задачи линейного программирования является оптимальным ее решением. Таким образом, для данной задачи решение $(8, 4, 0, 18, 0, 0)$ с целевым значением 28 является оптимальным. Теперь можно вернуться к исходной задаче линейного программирования, заданной уравнениями (29.53)–(29.57). Исходная задача содержит только переменные x_1, x_2 и x_3, поэтому оптимальное решение имеет вид $x_1 = 8$, $x_2 = 4$ и $x_3 = 0$ с целевым значением $(3 \cdot 8) + (1 \cdot 4) + (2 \cdot 0) = 28$. Заметим, что значения вспомогательных переменных в окончательном решении показывают, насколько велик резерв в каждом неравенстве. Вспомогательная переменная x_4 равна 18, а значение левой части в неравенстве (29.54) равно $8 + 4 + 0 = 12$, что на 18 меньше, чем значение правой части этого неравенства — 30. Вспомогательные переменные x_5 и x_6 равны нулю, и действительно, в неравенствах (29.55) и (29.56) левые и правые части равны. Обратите внимание на то,

что, даже если коэффициенты исходной канонической формы являются целочисленными, коэффициенты последующих эквивалентных задач не обязательно целочисленны, как не обязательно целочисленны и промежуточные решения. Более того, окончательное решение задачи линейного программирования также не обязательно будет целочисленным; в данном примере это не более чем совпадение.

Замещение

Формализуем процедуру замещения. Процедура PIVOT получает в качестве входных данных каноническую форму задачи линейного программирования, заданную кортежем (N, B, A, b, c, v), индекс l выводимой переменной x_l и индекс e вводимой переменной x_e. Она возвращает кортеж $(\widehat{N}, \widehat{B}, \widehat{A}, \widehat{b}, \widehat{c}, \widehat{v})$, описывающий новую каноническую форму. (Еще раз напомним, что элементы матриц A и $\widehat{A}$, имеющих размер $m \times n$, являются числами, обратными коэффициентам канонической формы.)

PIVOT(N, B, A, b, c, v, l, e)

```
1  // Вычисление коэффициентов уравнения для новой
   // базисной переменной x_e.
2  Пусть Â — новая матрица размером m × n
3  b̂_e = b_l / a_le
4  for каждого j ∈ N − {e}
5      â_ej = a_lj / a_le
6  â_el = 1 / a_le
7  // Вычисление коэффициентов остальных ограничений.
8  for каждого i ∈ B − {l}
9      b̂_i = b_i − a_ie b̂_e
10     for каждого j ∈ N − {e}
11         â_ij = a_ij − a_ie â_ej
12     â_il = −a_ie â_el
13 // Вычисление целевой функции.
14 v̂ = v + c_e b̂_e
15 for каждого j ∈ N − {e}
16     ĉ_j = c_j − c_e â_ej
17 ĉ_l = −c_e â_el
18 // Вычисление новых множеств базисных
   // и небазисных переменных.
19 N̂ = N − {e} ∪ {l}
20 B̂ = B − {l} ∪ {e}
21 return (N̂, B̂, Â, b̂, ĉ, v̂)
```

Процедура PIVOT работает следующим образом. В строках 3–6 вычисляются коэффициенты нового уравнения для x_e; для этого уравнение, в левой части которого стоит x_l, переписывается так, чтобы в левой части оказалась x_e. В строках 7–11 оставшиеся уравнения обновляются путем подстановки правой части

полученного нового уравнения вместо всех вхождений x_e. В строках 8–12 такая же подстановка выполняется для целевой функции, а в строках 14–17 обновляются множества небазисных и базисных переменных. Строка 21 возвращает новую каноническую форму. Если $a_{le} = 0$, вызов процедуры PIVOT приведет к ошибке (деление на нуль), однако, как будет показано в ходе доказательства лемм 29.2 и 29.12, данная процедура вызывается только тогда, когда $a_{le} \neq 0$.

Рассмотрим, как процедура PIVOT действует на значения переменных базисного решения.

Лемма 29.1

Рассмотрим вызов PIVOT(N, B, A, b, c, v, l, e), в котором $a_{le} \neq 0$. Пусть значения, возвращаемые вызовом, представляют собой $(\widehat{N}, \widehat{B}, \widehat{A}, \widehat{b}, \widehat{c}, \widehat{v})$ и пусть $\bar{x}$ обозначает базисное решение после выполнения вызова. Тогда

1. $\bar{x}_j = 0$ для каждого $j \in \widehat{N}$;
2. $\bar{x}_e = b_l / a_{le}$;
3. $\bar{x}_i = b_i - a_{ie}\widehat{b}_e$ для каждого $i \in \widehat{B} - \{e\}$.

Доказательство. Первое утверждение верно, поскольку в базисном решении все небазисные переменные задаются равными нулю. Если мы приравняем нулю все небазисные переменные в ограничении

$$x_i = \widehat{b}_i - \sum_{j \in \widehat{N}} \widehat{a}_{ij} x_j \ ,$$

то получим, что $\bar{x}_i = \widehat{b}_i$ для каждого $i \in \widehat{B}$. Поскольку $e \in \widehat{B}$, строка 3 процедуры PIVOT дает

$$\bar{x}_e = \widehat{b}_e = b_l / a_{le} \ ,$$

что доказывает второе утверждение. Аналогично, используя строку 9 для каждого $i \in \widehat{B} - \{e\}$, мы имеем

$$\bar{x}_i = \widehat{b}_i = b_i - a_{ie}\widehat{b}_e \ ,$$

что доказывает третье утверждение. ■

Формальный симплекс-алгоритм

Теперь мы готовы формализовать симплекс-алгоритм, который уже продемонстрировали на примере. Этот пример был очень удачным, однако еще необходимо ответить на следующие вопросы.

- Как определить, что задача линейного программирования является разрешимой?
- Что делать, если задача линейного программирования является разрешимой, однако начальное базисное решение не является допустимым?

- Как определить, что задача линейного программирования является неограниченной?
- Как выбирать вводимую и выводимую переменные?

В разделе 29.5 мы покажем, как определить, является ли задача разрешимой и, в случае положительного ответа, как найти каноническую форму, в которой начальное базисное решение является допустимым. На данном этапе предположим, что имеется процедура INITIALIZE-SIMPLEX(A, b, c), которая получает на входе задачу линейного программирования в стандартной форме, т.е. матрицу $A = (a_{ij})$ размером $m \times n$, m-мерный вектор $b = (b_i)$ и n-мерный вектор $c = (c_j)$. Если задача неразрешима, процедура возвращает соответствующее сообщение и завершается. В противном случае она возвращает каноническую форму, начальное базисное решение которой является допустимым.

Процедура SIMPLEX получает на входе задачу линейного программирования в стандартной форме, описанной выше. Она возвращает n-мерный вектор $\bar{x} = (\bar{x}_j)$, который является оптимальным решением задачи линейного программирования, заданной формулами (29.19)–(29.21).

```
SIMPLEX(A, b, c)
 1  (N, B, A, b, c, v) = INITIALIZE-SIMPLEX(A, b, c)
 2  Пусть Δ — новый вектор длиной m
 3  while c_j > 0 для некоторого индекса j ∈ N
 4      Выбрать индекс e ∈ N, для которого c_e > 0
 5      for каждого индекса i ∈ B
 6          if a_ie > 0
 7              Δ_i = b_i/a_ie
 8          else Δ_i = ∞
 9      Выбрать индекс l ∈ B, который минимизирует Δ_l
10      if Δ_l == ∞
11          return "задача неограниченная"
12      else (N, B, A, b, c, v) = PIVOT(N, B, A, b, c, v, l, e)
13  for i = 1 to n
14      if i ∈ B
15          x̄_i = b_i
16      else x̄_i = 0
17  return (x̄_1, x̄_2, ..., x̄_n)
```

Процедура SIMPLEX работает следующим образом. В строке 1 выполняется вызов упомянутой выше процедуры INITIALIZE-SIMPLEX(A, b, c), которая или определяет, что предложенная задача неразрешима, или возвращает каноническую форму, базисное решение которой является допустимым. Главная часть алгоритма содержится в цикле **while** в строках 3–12. Если все коэффициенты целевой функции отрицательны, цикл **while** завершается. В противном случае в строке 4 мы выбираем в качестве вводимой переменной некоторую переменную x_e, коэффициент при которой в целевой функции положителен. Хотя в качестве вводимой можно выбирать любую такую переменную, предполагается, что

используется некое предварительно заданное детерминистическое правило. Затем, в строках 5–9, выполняется проверка каждого ограничения и выбирается то, которое более всего лимитирует величину увеличения x_e, не приводящего к нарушению ограничений неотрицательности; базисная переменная, связанная с этим ограничением, выбирается в качестве выводимой переменной x_l. Если таких переменных несколько, можно выбрать любую из них, однако предполагается, что и здесь мы используем некое предварительно заданное детерминистическое правило. Если ни одно из ограничений не лимитирует возможность увеличения вводимой переменной, алгоритм выдает сообщение "задача неограниченная" (строка 11). В противном случае в строке 12 роли вводимой и выводимой переменных меняются путем вызова описанной выше процедуры PIVOT(N, B, A, b, c, v, l, e). В строках 13–16 вычисляется решение $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$ исходной задачи линейного программирования путем присваивания всем небазисным переменным нулевого значения, всем базисным переменным $\bar{x}_i$ — соответствующих значений b_i, а строка 17 возвращает эти значения.

Чтобы показать, что процедура SIMPLEX работает корректно, сначала покажем, что если в процедуре задано начальное допустимое значение и она завершилась, то при этом или возвращается допустимое решение, или сообщается, что данная задача является неограниченной. Затем мы покажем, что процедура SIMPLEX завершается; и наконец, в разделе 29.4 (теорема 29.10), будет показано, что возвращаемое процедурой решение является оптимальным.

Лемма 29.2

Пусть задана задача линейного программирования (A, b, c); предположим, что вызываемая в строке 1 процедуры SIMPLEX процедура INITIALIZE-SIMPLEX возвращает каноническую форму, базисное решение которой является допустимым. Тогда если процедура SIMPLEX возвращает решение в строке 17, это решение является допустимым решением задачи линейного программирования. Если процедура SIMPLEX выводит сообщение о неограниченности в строке 11, данная задача линейного программирования является неограниченной.

Доказательство. Воспользуемся следующим тройным инвариантом цикла:

в начале каждой итерации цикла **while** в строках 3–12

1. имеющаяся каноническая форма эквивалентна канонической форме, полученной в результате вызова процедуры INITIALIZE-SIMPLEX;
2. для всех $i \in B$ выполняется $b_i \geq 0$;
3. базисное решение, связанное с данной канонической формой, является допустимым.

Инициализация. Эквивалентность канонических форм для первой итерации очевидна. В формулировке леммы предполагается, что вызов процедуры INITIALIZE-SIMPLEX в строке 1 процедуры SIMPLEX возвращает каноническую форму, базисное решение которой является допустимым. Следовательно, третье утверждение справедливо. Далее, поскольку в базисном решении каж-

дой базисной переменной x_i присваивается значение b_i, а допустимость базисного решения предполагает неотрицательность всех базисных переменных x_i, то $b_i \geq 0$. Таким образом, второе утверждение инварианта также справедливо.

Сохранение. Покажем, что данный инвариант цикла сохраняется при условии, что оператор **return** в строке 11 не выполняется. Случай выполнения этой строки мы рассмотрим при обсуждении завершения цикла.

Каждая итерация цикла **while** меняет ролями некоторую базисную и небазисную переменные с помощью вызова процедуры PIVOT. Согласно упр. 29.3.3 новая каноническая форма эквивалентна канонической форме из предыдущей итерации, которая, согласно инварианту цикла, эквивалентна исходной канонической форме.

Теперь покажем, что сохраняется вторая часть инварианта цикла. Предположим, что в начале каждой итерации цикла **while** для всех $i \in B$ выполняется соотношение $b_i \geq 0$, и покажем, что эти неравенства остаются верными после вызова процедуры PIVOT в строке 12. Поскольку изменения в переменные b_i и множество B вносятся только в этой строке, достаточно показать, что она сохраняет данную часть инварианта. Пусть b_i, a_{ij} и B обозначают значения перед вызовом процедуры PIVOT, а $\widehat{b}_i$ — значения, возвращаемые процедурой PIVOT.

Во-первых, заметим, что $\widehat{b}_e \geq 0$, поскольку $b_l \geq 0$ согласно инварианту цикла, $a_{le} > 0$ согласно строкам 6 и 9 процедуры SIMPLEX, а $\widehat{b}_e = b_l/a_{le}$ согласно строке 3 процедуры PIVOT.

Для остальных индексов $i \in B - \{l\}$ имеем

$$\begin{aligned} \widehat{b}_i &= b_i - a_{ie}\widehat{b}_e && \text{(согласно строке 9 процедуры PIVOT)} \\ &= b_i - a_{ie}(b_l/a_{le}) && \text{(согласно строке 3 процедуры PIVOT)} \end{aligned} \tag{29.76}$$

Необходимо рассмотреть два случая: $a_{ie} > 0$ и $a_{ie} \leq 0$. Если $a_{ie} > 0$, то, поскольку l выбирается так, что

$$b_l/a_{le} \leq b_i/a_{ie} \quad \text{для всех } i \in B\ , \tag{29.77}$$

мы имеем

$$\begin{aligned} \widehat{b}_i &= b_i - a_{ie}(b_l/a_{le}) && \text{(согласно (29.76))} \\ &\geq b_i - a_{ie}(b_i/a_{ie}) && \text{(согласно (29.77))} \\ &= b_i - b_i \\ &= 0\ , \end{aligned}$$

и, таким образом, $\widehat{b}_i \geq 0$. Если $a_{ie} \leq 0$, то, поскольку a_{le}, b_i и b_l неотрицательны, из уравнения (29.76) вытекает, что неотрицательным должно быть и значение $\widehat{b}_i$.

Теперь докажем, что это базисное решение является допустимым, т.е. что все переменные имеют неотрицательные значения. Небазисные переменные устанавливаются равными нулю и, следовательно, являются неотрицательными. Каждая базисная переменная x_i задается уравнением

$$x_i = b_i - \sum_{j \in N} a_{ij} x_j \ .$$

В базисном решении $\bar{x}_i = b_i$. Используя вторую часть инварианта цикла, можно сделать вывод, что все базисные переменные $\bar{x}_i$ неотрицательны.

Завершение. Цикл **while** может завершиться одним из двух способов. Если его завершение связано с выполнением условия в строке 3, то текущее базисное решение является допустимым и это решение возвращается строкой 17. Второй способ завершения связан с возвращением сообщения "задача неограниченная" строкой 11. В этом случае в каждой итерации цикла **for** (строки 5–8) при выполнении строки 6 оказывается, что $a_{ie} \leq 0$. Рассмотрим решение $\bar{x}$, определяемое следующим образом:

$$\bar{x}_i = \begin{cases} \infty \ , & \text{если } i = e \ , \\ 0 \ , & \text{если } i \in N - \{e\} \ , \\ b_i - \sum_{j \in N} a_{ij} \bar{x}_j \ , & \text{если } i \in B \ . \end{cases}$$

Покажем, что данное решение является допустимым, т.е. что все переменные являются неотрицательными. Небазисные переменные, отличные от $\bar{x}_e$, равны нулю, а значение $\bar{x}_e = \infty > 0$, следовательно, все небазисные переменные неотрицательны. Для каждой базисной переменной $\bar{x}_i$ имеем

$$\begin{aligned} \bar{x}_i &= b_i - \sum_{j \in N} a_{ij} \bar{x}_j \\ &= b_i - a_{ie} \bar{x}_e \ . \end{aligned}$$

Из инварианта цикла вытекает, что $b_i \geq 0$, и мы имеем $a_{ie} \leq 0$ и $\bar{x}_e = \infty > 0$. Таким образом, $\bar{x}_i \geq 0$.

Теперь покажем, что целевое значение этого решения $\bar{x}$ является неограниченным. Из уравнения (29.42) целевое значение равно

$$\begin{aligned} z &= v + \sum_{j \in N} c_j \bar{x}_j \\ &= v + c_e \bar{x}_e \ . \end{aligned}$$

Поскольку $c_e > 0$ (согласно строке 4 процедуры SIMPLEX) и $\bar{x}_e = \infty$, целевое значение бесконечно, а значит, задача линейного программирования неограниченная. ■

Остается показать, что процедура SIMPLEX завершается и что после ее завершения возвращаемое решение является оптимальным. Оптимальность будет рассмотрена в разделе 29.4. Сейчас же мы рассмотрим завершение процедуры.

Завершение

В приведенном в начале данного раздела примере каждая итерация симплекс-алгоритма увеличивала целевое значение, связанное с базисным решением. В упр. 29.3.2 предлагается показать, что ни одна итерация процедуры SIMPLEX не может уменьшить целевое значение, связанное с базисным решением. К сожалению, может оказаться, что итерация оставляет целевое значение неизменным. Это явление называется ***вырожденностью***, и сейчас мы рассмотрим его более подробно.

Целевое значение изменяется в строке 14 процедуры PIVOT в результате присваивания $\widehat{v} = v + c_e\widehat{b}_e$. Поскольку вызов процедуры PIVOT в процедуре SIMPLEX происходит только при $c_e > 0$, целевое значение может остаться неизменным (т.е. $\widehat{v} = v$) только в том случае, когда $\widehat{b}_e$ будет равно нулю. Это значение вычисляется как $\widehat{b}_e = b_l/a_{le}$ в строке 3 процедуры PIVOT. Поскольку процедура PIVOT вызывается только при $a_{le} \neq 0$, для того, чтобы $\widehat{b}_e$ было равно нулю и, как следствие, целевое значение осталось неизменным, должно выполняться условие $b_l = 0$.

Такая ситуация действительно может возникнуть. Рассмотрим следующую задачу линейного программирования:

$$\begin{aligned} z &= x_1 + x_2 + x_3 \\ x_4 &= 8 - x_1 - x_2 \\ x_5 &= x_2 - x_3 \,. \end{aligned}$$

Предположим, что в качестве вводимой переменной выбрана переменная x_1, а в качестве выводимой — x_4. После замещения получим следующую задачу:

$$\begin{aligned} z &= 8 + x_3 - x_4 \\ x_1 &= 8 - x_2 - x_4 \\ x_5 &= x_2 - x_3 \,. \end{aligned}$$

В этой ситуации единственная возможность замещения — когда вводимой переменной является x_3, а выводимой переменной — x_5. Поскольку $b_5 = 0$, после замещения целевое значение 8 останется неизменным:

$$\begin{aligned} z &= 8 + x_2 - x_4 - x_5 \\ x_1 &= 8 - x_2 - x_4 \\ x_3 &= x_2 - x_5 \,. \end{aligned}$$

Целевое значение осталось неизменным, но представление задачи изменилось. К счастью, если мы продолжим замещение, выбрав в качестве вводимой переменной x_2, а в качестве выводимой — x_1, целевое значение увеличится (до 16) и симплекс-алгоритм сможет продолжить свою работу.

Вырожденность является единственным возможным препятствием для окончания симплекс-алгоритма, так как может привести к явлению, именуемому ***зацикливанием***, когда канонические формы на двух разных итерациях одинаковы. Из-за вырожденности процедура SIMPLEX может выбирать последовательность замещающих операций, которые оставляют целевое значение неизменным, но при этом повторяют последовательность из одних и тех же канонических форм. Поскольку алгоритм SIMPLEX детерминированный, в случае зацикливания он бесконечно проходит по одной и той же последовательности канонических форм, никогда не заканчивая свою работу.

Зацикливание — единственная причина, по которой процедура SIMPLEX может не завершить свою работу. Чтобы показать это, сначала разработаем некоторую дополнительную технику.

На каждой итерации процедура SIMPLEX в дополнение к множествам N и B поддерживает A, b, c и v. Хотя для эффективной реализации симплекс-алгоритма требуется явная поддержка A, b, c и v, можно обойтись и без нее. Другими словами, множества базисных и небазисных переменных достаточно для того, чтобы единственным образом определить каноническую форму. Перед тем как доказать этот факт, докажем одну полезную алгебраическую лемму.

Лемма 29.3

Пусть I представляет собой множество индексов и пусть для каждого $j \in I$ α_j и β_j — действительные числа, а x_j — действительная переменная. Пусть также γ — некоторое действительное число. Предположим, что для любых x_j выполняется следующее условие:

$$\sum_{j \in I} \alpha_j x_j = \gamma + \sum_{j \in I} \beta_j x_j \ . \tag{29.78}$$

Тогда $\alpha_j = \beta_j$ для каждого $j \in I$, и $\gamma = 0$.

Доказательство. Поскольку уравнение (29.78) выполняется для любых значений x_j, можно выбрать для них определенные значения, чтобы сделать заключения об α, β и γ. Выбрав $x_j = 0$ для всех $j \in I$, можно сделать вывод, что $\gamma = 0$. Теперь выберем произвольный индекс $j \in I$ и зададим $x_j = 1$ и $x_k = 0$ для всех $k \neq j$. В таком случае должно выполняться равенство $\alpha_j = \beta_j$. Поскольку индекс j выбирался из множества I произвольным образом, можно заключить, что $\alpha_j = \beta_j$ для всех $j \in I$. ■

Конкретная задача линейного программирования может иметь много различных канонических форм; вспомним, что любая каноническая форма имеет то же самое множество допустимых и оптимальных решений, что и исходная задача. Теперь покажем, что каноническая форма любой задачи линейного программирования уникальным образом определяется множеством базисных переменных, т.е. что с заданным набором базисных переменных связана единственная каноническая форма (с однозначно определяемым множеством коэффициентов в правой части).

Лемма 29.4

Пусть (A, b, c) представляет собой задачу линейного программирования в стандартной форме. Задание множества базисных переменных B однозначно определяет соответствующую каноническую форму.

Доказательство. Будем проводить доказательство от противного. Предположим, что существуют две различные канонические формы с одинаковым множеством базисных переменных B. Эти канонические формы должны также иметь одинаковые множества небазисных переменных $N = \{1, 2, \ldots, n+m\} - B$. Запишем первую каноническую форму как

$$z = v + \sum_{j \in N} c_j x_j \tag{29.79}$$

$$x_i = b_i - \sum_{j \in N} a_{ij} x_j \quad \text{для } i \in B\,, \tag{29.80}$$

а вторую как

$$z = v' + \sum_{j \in N} c'_j x_j \tag{29.81}$$

$$x_i = b'_i - \sum_{j \in N} a'_{ij} x_j \quad \text{для } i \in B\,. \tag{29.82}$$

Рассмотрим систему уравнений, образованную путем вычитания каждого уравнения строки (29.82) из соответствующего уравнения строки (29.80). Полученная система имеет вид

$$0 = (b_i - b'_i) - \sum_{j \in N} (a_{ij} - a'_{ij}) x_j \quad \text{для } i \in B$$

или, что эквивалентно,

$$\sum_{j \in N} a_{ij} x_j = (b_i - b'_i) + \sum_{j \in N} a'_{ij} x_j \quad \text{для } i \in B\,.$$

Теперь для всех $i \in B$ применим лемму 29.3, где $\alpha_j = a_{ij}$, $\beta_j = a'_{ij}$, $\gamma = b_i - b'_i$ и $I = N$. Поскольку $\alpha_j = \beta_j$, имеем $a_{ij} = a'_{ij}$ для всех $j \in N$, а поскольку $\gamma = 0$, то $b_i = b'_i$. Таким образом, в этих двух канонических формах матрицы A и A' и векторы b и b' идентичны. С помощью аналогичных рассуждений в упр. 29.3.1 показано, что в этом случае также справедливо $c = c'$ и $v = v'$; следовательно, рассматриваемые канонические формы должны быть идентичны. ■

Теперь можно показать, что зацикливание — единственная возможная причина, по которой процедура SIMPLEX может не завершиться.

Лемма 29.5

Если процедура SIMPLEX не завершается не более чем за $\binom{n+m}{m}$ итераций, она зацикливается.

Доказательство. Согласно лемме 29.4 множество базисных переменных B однозначно определяет каноническую форму. Всего имеется $n + m$ переменных, а $|B| = m$, так что существует не более чем $\binom{n+m}{m}$ способов выбрать B. Следовательно, всего имеется не более чем $\binom{n+m}{m}$ различных канонических форм. Поэтому, если процедура SIMPLEX совершает более чем $\binom{n+m}{m}$ итераций, она должна зациклиться. ■

Зацикливание теоретически возможно, но встречается чрезвычайно редко. Его можно избежать путем несколько более аккуратного выбора вводимых и выводимых переменных. Один из способов состоит в подаче на вход слабого возмущения, что приводит к невозможности получить два решения с одинаковым целевым значением. Второй способ заключается в том, чтобы всегда выбирать переменную с наименьшим индексом. Эта стратегия известна как ***правило Бленда*** (Bland's rule). Мы не будем приводить доказательства, что эти стратегии позволяют избежать зацикливания.

Лемма 29.6

Если в строках 4 и 9 процедуры SIMPLEX всегда выполняется выбор переменной с наименьшим индексом, процедура SIMPLEX должна завершиться. ■

Мы завершим данный раздел следующей леммой.

Лемма 29.7

Если процедура INITIALIZE-SIMPLEX возвращает каноническую форму, базисное решение которой является допустимым, то процедура SIMPLEX или выдает сообщение о неограниченности задачи линейного программирования, или завершается возвратом допустимого решения не более чем за $\binom{n+m}{m}$ итераций.

Доказательство. В леммах 29.2 и 29.6 показано, что если процедура INITIALIZE-SIMPLEX возвращает каноническую форму, базисное решение которой является допустимым, то процедура SIMPLEX либо выдает сообщение о неограниченности задачи линейного программирования, либо завершается, предоставляя допустимое решение. Используя обращение леммы 29.5, приходим к заключению, что если процедура SIMPLEX завершается предоставлением допустимого решения, то это происходит не более чем за $\binom{n+m}{m}$ итераций. ■

Упражнения

29.3.1

Завершите доказательство леммы 29.4, показав, что $c = c'$ и $v = v'$.