Лекции по комбинаторной оптимизации

©2018 М.А. Посыпкин
(119333 Москва, ул. Вавилова, 40, ВЦ ФИЦ ИУ РАН)
e-mail: mposypkin@gmail.com

Оглавление

Глава 1

Теория вычислений

1.1 Языки

Определение 1 ***Алфавитом*** *называется любой конечный набор символов.*

Определение 2 ***Языком*** *называется в алфавите Σ называется любое подмножество L множества Σ^* всех конечных комбинаций символов Σ.*

1.2 Машина Тьюринга

1.2.1 Детерминированная машина Тьюринга

Машина Тьюринга предложена Аланом Тьюрингом для формализации понятия алгоритма. Формальное определение понятия алгоритма дает возможность доказывать различные результаты из области теорий сложности и вычислимости.

Детерминированная машина Тьюринга (ДМТ) представляет собой конечный автомат с лентой, в ячейках которой записываются символы конечного алфавита Г. Лента является бесконечной в обе стороны, а ее ячейки пронумерованы целыми числами. **Считывающая/записывающая головка** (далее головка) машины Тьюринга указывает на одну из позиций в ленте.

Программа для детерминированной машины Тьюринга (ДМТ-программа) определяется конечным множеством состояний Q с тремя выделенными состояниями $\{q_0, q_Y, q_N\}$, где q_0 — начальное состояния, q_Y — конечное "положительное" состояние, q_N — конечное "отрицательное" состояние.

Функция перехода $\delta(q, s) = (q', s', \Delta)$ задает шаг машины Тьюринга, при условии, что головка считала символ s, а автомат находился в состоянии q. Функция перехода ставит в соответствие паре (q, s), где q — любое состояние из Q кроме q_Y, q_N, а s —любой символ алфавита Γ тройку (q', s', Δ). При этом q' — новое состояние, в которое перейдет автомат и символ s', который будет записан вместо символа s. Третий элемент Δ определяет перемещение вперед, если $\Delta = 1$ и назад, если $\Delta = -1$. Если $\Delta = 0$, то перемещения не производится.

Перед началом выполнения программы, машина Тьюринга находится в состоянии q_0, а в ячейках с номерами $1, 2, \ldots, n$ записано входное слово x, $|x| = n$. Головка указывает на ячейку с номером 0. Все другие ячейки пустые, т.е. содержат специальный символ B.

Корректная ДМТ-программа должна на любом входном слове завершать свою работу за конечное число шагов в одном из конечных состояний q_Y — *слово принято* или q_N — *слово не принято.*

Пример 1 Пример машины Тьюринга, решающей, содержит ли двоичное число четное количество нулей.

Состояния: q_0, q_1, q_Y, q_N

	0	1	B
q_0	$(q_1, 0, 1)$	$(q_0, 1, 1)$	$(q_Y, B, 0)$
q_1	$(q_0, 0, 1)$	$(q_1, 1, 1)$	$(q_N, B, 0)$

Определение 3 Язык $L \subseteq \Gamma^*$ ***распознается программой*** Π*, если он содержит только те слова из* Σ^**, которые принимаются программой* Π*, т.е. программа завершается в состоянии* q_Y *на них.*

Далее будем рассматривать только такие программы, которые останавливаются на любом слове из Γ^*.

Упражнение 1 Написать программу для детерминированной машины Тьюринга, которая прибавляет единицу к двоичному (десятичному, троичному и т.п.) целому числу на ленте.

Упражнение 2 На ленте машины Тьюринга содержится двоичная последовательность. Напишите программу для машины Тьюринга, которая вычисляет двоичное дополнение записанного числа, т.е. заменяет каждый символ 1 на 0 и наоборот.

1.2.2 Недетерминированная машина Тьюринга

Работа недетерминированной машины Тьюринга делится на две фазы: фаза угадывания и рабочая фаза. Сначала на ленте записываются данные задачи x в заданной схеме кодирования с использованием определенного алфавита. Затем, слева от $x \in \Gamma^*$ на ленте угадывающее устройство записывает некоторое количество символов гипотезы $y \in \Gamma^*$. После чего начинается работа программы обычной детерминированной машины Тьюринга, которая проверяет, является ли гипотеза y ответом для решаемой задачи.

Сложность работы фазы угадывания полагается равной сложности записи гипотезы, т.е. ее длине $|y|$.

Будем говорить, что вход x *принимается* программой для НДМТ, если найдется такая гипотеза y, что в результате выполнения недетерминированного вычисления машина приходит в состояние q_Y.

1.3 Понятие классов P и NP

Ограничимся рассмотрением **задач распознавания**, ответ на которые может быть двух типов: **ДА** либо **НЕТ**. Язык $L(I)$, соответствующий некоторой задаче I, состоит из тех последовательностей, которые соответствуют задачам, ответ на которые — **ДА**.

Пусть Π — программа для детерминированной машины Тьюринга, работающая на входах из алфавита Σ. Обозначим через $s_\Pi(x)$ число шагов (переходов) машины Тьюринга на слове $x \in \Sigma^*$.

Определение 4 ***Сложностью ДМТ-программы*** Π *называется функция* $S_\Pi(n) = \{\max m : \exists x \in \Sigma^*, |x| = n, s_\Pi(x) = m\}$.

Определение 5 ДМТ-программа Π *называется* ***полиномиальной****, если существует полином* $p(n)$ *степени* k*, такой что* $S_\Pi(n) \leq p(n)$ *при любом натуральном* n*.*

Данное определение более компактно может быть записано формулой $p(n) = O(n^k)$.

Определение 6 Язык, распознаваемый полиномиальной программой называется ***полиномиальным****.*

Определение 7 ***Сложностью проверки принимаемого входа*** x $\tilde{s}_\Pi(x)$ *НДМТ-программой* Π *для НДМТ будем называть минимальное число шагов по всем возможным гипотезам* y*, требуемое для получения* q_Y*.*

Определение 8 ***Сложностью НДМТ-программы*** Π *называется функция* $\tilde{S}_\Pi(n) = \{\max 1 \cup \{m : \exists x \in \Sigma^*, |x| = n, \tilde{s}_\Pi(x) = m\}\}$.

Таким образом, если нет ни одного входа длиной n, принимаемого НДМТ-программой, то ее сложность для этого n полагается равной 1.

По аналогии определяется понятие полиномиальной НДМТ-программы.

Определение 9 НДМТ-программа Π *называется* ***полиномиальной****, если существует полином* $p(n)$ *степени* k*, такой что* $\tilde{S}_\Pi(n) \leq p(n)$ *при любом натуральном* n.

Класс P состоит из всех задач, для которых существует полиномиальная ДМТ-программа, класс NP — из всех задач, для которых которых существует полиномиальная НДМТ-программа.

Вопрос о равенстве классов P и NP на данный момент открыт, хотя есть веские основания полагать, что эти два класса не совпадают. Справедлива следующая теорема.

Теорема 1 Если задача принадлежит классу NP, то существует решающая ее ДМТ-программа Π*, сложность работы которой ограничена сверху величиной* $2^{q(n)}$*, где* $q(n)$ *— некоторый полином.*

Доказательство. Рассмотрим НДМТ-программу $\tilde{\Pi}$, решающую заданную задачу. Сложность ее ограничена сверху некоторым полиномом $p(n)$. Следовательно, гипотеза имеет длину не более $p(n)$. Все множество гипотез имеет мощность не более $|\Sigma|^{p(n)}$. Построим ДМТ-программу Π, которая последовательно выписывает все гипотезы и запускает для каждой из них алгоритм проверки из программы $\tilde{\Pi}$. Сложность программы Π не превосходит величины $|\Sigma|^{p(n)}p(n)$. Очевидно, что существует такой полином $q(n)$, что $S_\Pi(n) \leq 2^{q(n)}$.

1.4 Полиномиальная сводимость и NP-полные задачи

1.4.1 Полиномиальная сводимость

Определение 10 Говорят, что язык L_1 ***полиномиально сводится к языку*** L_2 *(*$L_1 \propto L_2$*), если существует полиномиальная ДМТ-программа* Π*, осуществляющая отображение* $f : \Sigma^* \to \Sigma^*$*, такое что* $x \in L_1$ *тогда и только тогда, когда* $f(x) \in L_2$.

Очевидно, что определенное отношение сводимости транзитивно. Также очевидна справедливость следующего утверждения.

Утверждение 1 Если $L_2 \in P$ и $L_1 \propto L_2$, то $L_1 \in P$.

*Определение 11 Задача называется **NP-полной**, если любая другая задача из класса NP может быть к ней полиномиально сведена.*

1.4.2 Теорема Кука

Исторически первой NP-полной задачей является задача о выполнимости булевой формулы, формулируемая следующим образом.

Задача о выполнимости. Задан набор булевых переменных $x_1, \ldots, x_n$ и формула вида

$f(x_1, \ldots, x_n) = \wedge_{j=1}^{m} K_j$, где $K_j = \vee_{i=1}^{n} s_i^j$, $s_i^j \in \{x_i, \overline{x_i}, 0\}$.

Задача о выполнимости состоит в том, чтобы найти такой набор $\alpha_1, \ldots, \alpha_n$ значений переменных $x_1, \ldots, x_n$, что $f(\alpha_1, \ldots, \alpha_n) = 1$. Если такой набор существует, то будем говорить, что данная задача **разрешима**.

*Теорема 2 **(Теорема Кука)** Любой язык класса NP полиномиально сводим к языку, порождаемому задачей о выполнимости.*

Доказательство. Пусть язык L относится к классу NP. Тогда существует полиномиальная НДМТ-программа $\Pi = [\Sigma, Q, \delta]$, порождающая этот язык. Рассмотрим слово $x, |x| = n$, принимаемое данной программой. Для доказательства теоремы достаточно построить полиномиальный алгоритм, порождающий по слову x задачу о выполнимости $sat(x)$, разрешимую в том и только том случае, когда $x \in L$.

Пронумеруем все состояния из множества Q таким образом, что состояния q_0, q_Y и q_N имеют номера $0, 1, 2$ соответственно. Символы из алфавита Σ пронумеруем таким образом, чтобы пробельный символ T имел номер 0. Заметим, что число ячеек ленты, на которые показывает головка машины Тьюринга в процессе работы, лежит в интервале $[-p(n), p(n)+1]$. Введем набор переменных $H_{i,j}$, таких что $H_{i,j} = 1$, если считывающее устройство находится в ячейке j на шаге i, $0 \le i \le p(n)$, $-p(n) \le j \le p(n)+1$. Также введем переменные $Q_{i,j}$, $0 \le i \le p(n)$, $0 \le j \le |Q|$, принимающие значение 1, если на шаге i машина находится в состоянии q_j, 0 в противном случае. Определим переменные $S_{i,j,k}$, принимающие значение 1, если на шаге i в j-й ячейке ленты записан символ с номером k в алфавите Σ.

Определим несколько булевых формул, одновременное выполнение которых определяет корректное выполнение ДМТ-программы.

Первая формула определяет то, что перед началом вычислений машина находится в начальном состоянии, головка показывает на позицию с номером 1, а в ячейках с номерами $1, \ldots, |x|$ записано слово x.

$$\Phi_1 = Q_{0,0} \wedge H_{0,1} \wedge \wedge_{i=1}^{|x|} S_{0,i,\nu(x_i)} \wedge \wedge_{i=|x|+1}^{p(n)} S_{0,i,0},$$

где $\nu(x_i)$ — номер i-го символа слова x.

Вторая формула кодирует утверждение "На любом шаге ДМТ-программа находится только в одном состоянии".

$$\Phi_2 = \wedge_{i=0}^{p(n)} \left(\left(\vee_{j=0}^{|Q|-1} Q_{i,j} \right) \wedge \left(\wedge_{0 \le j < j' \le |Q|-1} \neg \left(Q_{i,j} \wedge Q_{i,j'} \right) \right) \right),$$

Данная формула также легко записывается в виде КНФ:

$$\Phi_2 = \wedge_{i=0}^{p(n)} \left(\vee_{j=0}^{|Q|-1} Q_{i,j} \right) \wedge \wedge_{i=0}^{p(n)} \left(\wedge_{0 \le j < j' \le |Q|-1} \left(\neg Q_{i,j} \vee \neg Q_{i,j'} \right) \right).$$

Третья формула определяет то, что на каждом шаге считывающее устройство находится только в одной из допустимых ячеек. Она формируется по-аналогии с предыдущей группой с использованием переменных $H_{i,j}$:

$$\Phi_3 = \wedge_{i=0}^{p(n)} \left(\left(\vee_{j=-p(n)}^{p(n)+1} H_{i,j} \right) \wedge \left(\wedge_{-p(n) \le j < j' \le p(n)+1} \neg \left(H_{i,j} \wedge H_{i,j'} \right) \right) \right),$$

Данную формулу также перепишем в виде КНФ:

$$\Phi_3 = \wedge_{i=0}^{p(n)} \left(\vee_{j=-p(n)}^{p(n)+1} H_{i,j} \right) \wedge \wedge_{i=0}^{p(n)} \left(\wedge_{-p(n) \le j < j' \le p(n)+1} \left(\neg H_{i,j} \vee \neg H_{i,j'} \right) \right),$$

Четвертая формула задает то, что на каждом шаге в каждой ячейке записан только один символ алфавита Σ:

$$\Phi_4 = \wedge_{i=0}^{p(n)} \wedge_{j=-p(n)}^{p(n)+1} \left(\left(\vee_{k=0}^{|\Sigma|-1} S_{i,j,k} \right) \wedge \left(\wedge_{0 \le k < k' \le |\Sigma|-1} \neg \left(S_{i,j,k} \wedge S_{i,j,k'} \right) \right) \right),$$

Вариант в КНФ выглядит так:

$$\Phi_4 = \wedge_{i=0}^{p(n)} \wedge_{j=-p(n)}^{p(n)+1} \left(\vee_{k=0}^{|\Sigma|-1} S_{i,j,k} \right) \wedge \wedge_{i=0}^{p(n)} \wedge_{j=-p(n)}^{p(n)+1} \left(\wedge_{0 \le k < k' \le |\Sigma|-1} \left(\neg S_{i,j,k} \vee \neg S_{i,j,k'} \right) \right),$$

Пятая формула гарантирует, что если головка не находится в позиции j на шаге i, то символ в ней на шаге $i+1$ останется прежним:

$$\Phi_5 = \wedge_{i=1}^{p(n)-1} \wedge_{j=-p(n)}^{p(n)+1} \wedge_{k=0}^{|\Sigma|-1} \left(S_{i,j,k} \wedge \neg H_{i,j} \Rightarrow S_{i+1,j,k} \right).$$

Воспользовавшись тем, что формула $A \Rightarrow B$ равносильна формуле $\neg A \vee B$, получим:

$$\Phi_5 = \wedge_{i=1}^{p(n)-1} \wedge_{j=-p(n)}^{p(n)+1} \wedge_{k=0}^{|\Sigma|-1} (\neg S_{i,j,k} \vee H_{i,j} \vee S_{i+1,j,k}) .$$

Шестая формула задает выполнение переходов программы в соответствии с функцией переходов δ.

$$\Phi_6 = \wedge_{i=0}^{p(n)-1} \wedge_{j=-p(n)}^{p(n)+1} \wedge_{k=0}^{|\Sigma|-1} \wedge_{l=0}^{q} (H_{i,j} \wedge S_{i,j,k} \wedge Q_{i,l} \Rightarrow H_{i+1,j+\Delta} \wedge Q_{i+1,l'} \wedge S_{i+1,j,k'}) , \tag{1.1}$$

где числа l', k', Δ определяются следующим образом:

$$\delta(q_l, s_k) = (q'_l, s'_k, \Delta), \text{ если } q_l \in Q \setminus \{q_Y, q_N\},$$
$$l = l', k = k', \Delta = 0, \text{ если } q_l \in \{q_Y, q_N\}.$$

Данная формула может быть представлена в виде КНФ (упражнение 3).

Последняя формула определяет тот факт, что вычисления остановятся не позднее, чем через $p(n)$ шагов.

$$\Phi_7 = Q_{p(n),i}.$$

Формула в построенной задаче о выполнимости имеет вид $\Phi = \Phi_1 \wedge \cdots \wedge \Phi_7$. Формулы $\Phi_1, \ldots, \Phi_7$ строились таким образом, что если найдется такой набор значений переменных, что на нем формула Φ принимает значение 1, то существует гипотеза y, на которой НДМТ-программа остановится в состоянии q_Y не более чем через $p(n)$ шагов. И наоборот, если существует гипотеза y, на которой НДМТ-программа остановится в состоянии q_Y, то формула Φ — выполнима.

Несложно показать (упражнение 4), что число переменных и операций в формулах $\Phi_1, \ldots, \Phi_7$ ограничено сверху полиномом от n. Из этого следует, что рассматриваемая задача полиномиально сводима к задаче выполнимости. Тем самым теорема Кука доказана. □

Упражнение 3 Приведите формулу (1.1) *к КНФ.*

Упражнение 4 Покажите, что число переменных и операций в формулах $\Phi_1, \ldots, \Phi_7$ *в доказательстве теоремы Кука ограничено полиномом от* n.

1.5 Примеры NP-полных задач

Очевидно, что любая задача, к которой полиномиально сводится другая NP-полная задача, также является NP-полной. Этот факт удобно использовать для установления NP-полноты различных задач комбинаторной оптимизации.

1.5.1 Задача о 3-выполнимости

Известной NP-полной задачей является задача **3-выполнимости** (3-SAT).

*Определение 12 Коньюнктивная нормальная форма, в которой каждая дизъюнкция состоит в точности из трех литералов, называется **3-КНФ**.*

Задача о выполнимости формулируется следующим образом: дана булева 3-КНФ формула, требуется определить, существует ли набор, на котором функция, задаваемая этой формулой, принимает значение 1.

Утверждение 2 Задача о 3-выполнимости является NP-полной.

Доказательство. Для того, чтобы доказать данное утверждение достаточно показать, что задача о выполнимости в произвольной форме полиномиально сводима к задаче о 3-выполнимости. Покажем, что любая дизъюнкция литералов $\Phi = \lambda_1 \vee \cdots \vee \lambda_k$ может быть за полиномиальное время сведена к 3-КНФ Φ', такой что для Φ существует выполняющий набор тогда и только тогда, когда таковой сущестует для Φ'. При этом Φ' может содержать новые переменные по отношению к Φ. Рассмотрим четыре возможных варианта:

1. Выполнено $k = 1$. Тогда соответствующая 3-КНФ имеет вид

$$(\lambda_1 \vee y_1 \vee y_2) \wedge (\lambda_1 \vee \neg y_1 \vee y_2) \wedge (\lambda_1 \vee y_1 \vee \neg y_2) \wedge (\lambda_1 \vee \neg y_1 \vee \neg y_2).$$

2. Выполнено $k = 2$. Тогда соответствующая 3-КНФ имеет вид

$$(\lambda_1 \vee \lambda_2 \vee y_1) \wedge (\lambda_1 \vee \lambda_2 \vee \neg y_1).$$

3. Выполнено $k = 3$. В этом случае 3-КНФ совпдает с дизъюнкцией.

4. Выполнено $k > 3$. Заметим, что для булевой формулы $\Phi = \Phi_1 \vee \Phi_2$ существует выполняющий набор тогда и только тогда, когда таковой существует для $(\Phi 1 \vee y_1) \wedge (\neg y_1 \vee \Phi_2)$. Последовательно вводя переменные $y_1, y_2, \ldots, y_{k-3}$ и применяя данное правило, получим следующую цепочку формул:

$$\begin{aligned}
\Phi &= \lambda_1 \vee \lambda_2 \vee \cdots \vee \lambda_k, \\
\Phi_1 &= (\lambda_1 \vee \lambda_2 \vee y_1) \wedge (\neg y_1 \vee \lambda_3 \vee \cdots \vee \lambda_k), \\
&(\lambda_1 \vee \lambda_2 \vee y_1) \wedge (\neg y_1 \vee \lambda_3 \vee y_2) \wedge (\neg y_2 \vee \cdots \vee \lambda_k), \\
&\cdots \\
\Phi_{k-3} &= (\lambda_1 \vee \lambda_2 \vee y_1) \wedge (\neg y_1 \vee \lambda_3 \vee y_2) \wedge \cdots \wedge (\neg y_{k-3} \vee \lambda_{k-1} \vee \lambda_k).
\end{aligned}$$

Положим $\Phi' = \Phi_3$. Очевидно, Φ' является 3-КНФ и, по построению, для Φ существует выполняющий набор тогда и только тогда, когда таковой сущестует для Φ'. Предложенная схема трансляции является полиномиальной по общему числу литералов в исходной конъюнкции. Тем самым, теорема доказана. $\square$

Упражнение 5 Свести КНФ к 3-КНФ, записать последовательность преобразований.

1. $(\overline{x_1} \vee x_2) \wedge (x_1 \vee x_2 \vee x_3 \vee \overline{x_4})$,
2. $\overline{x_1}$,
3. $\overline{x_1} \vee x_2$.

1.5.2 Задача о трехмерном сочетании

Сформулируем задачу о трехмерном сочетании. Задано подмножество M множества $X \times Y \times Z$, где X, Y, Z — три конечных множества, каждое из которых содержит одинаковое число элементов q. Требуется выяснить, содержит ли множество M такое подмножество M' мощности q, что любые два его различных элемента отличаются во всех трех компонентах. Другими словами для любых $u, v \in M', u \neq v$ справедливо $u_1 \neq v_1, u_2 \neq v_2, u_3 \neq v_3$.

Подобное подмножество, если оно существует будем называть 3-**сочетанием**.

Утверждение 3 Задача о трехмерном сочетании является NP-полной.

Доказательство. То, что рассматриваемая задача относится к классу NP, следует из того, что проверка правильности найденного (угаданного НДМТ) решения очевидно может быть выполнена за полиномиальное время. Поэтому для доказательсва утверждения достаточно показать, что задача 3-выполнимости полиномиально сводится к задаче о трехмерном сочетании.

Пусть дана задача 3-выполнимости, формула в которой имеет n переменных $x_1, \ldots, x_n$ и m дизъюнкций $d_1, \ldots, d_m$.

Для каждой переменной x_i, $i = 1, \ldots, n$ построим множества

$$X_i = \{x_i^1, \overline{x}_i^1, x_i^2, \overline{x}_i^2, \ldots, x_i^m, \overline{x}_i^m\},\ |X_i| = 2m,$$
$$A_i = \{a_i^1, a_i^2, \ldots, a_i^m\},\ |A_i| = m,$$
$$B_i = \{b_i^1, b_i^2, \ldots, b_i^m\},\ |B_i| = m.$$

Составим множества

$$T_i^t = \{(\overline{x}_i^j, a_i^j, b_i^j),\ j = 1, \ldots, m\},\ |T_i^t| = m$$
$$T_i^f = \{(x_i^j, a_i^{j'}, b_i^j),\ j = 1, \ldots, m\},\ |T_i^f| = m$$
$$\text{где } j' = \begin{cases} j+1, \text{ если } j < m, \\ 1, \text{ если } j = m. \end{cases}$$

Положим $T_i = T_i^t \cup T_i^f$, $T = \cup_{i=1,\ldots,n} T_i$. Очевидно, $|T| = 2mn$.

Заметим, что j-я тройка из множества T_i^t имеет общие компоненты с тройками с индексами j и j'' из T_i^f, где

$$j'' = \begin{cases} j-1, \text{ если } j > 1, \\ m, \text{ если } j = 1. \end{cases}$$

Также j-я тройка из T_i^f имеет общие компоненты с тройками с индексами j' и j из T_i^t.

Определим множества

$$U = \{u_1, u_2, \ldots, u_m\},\ |U| = m,$$
$$V = \{v_1, v_2, \ldots, v_m\},\ |V| = m.$$

Для каждой дизъюнкции d_i, $i = 1, \ldots, m$, определим множество троек $D_j = \{(x_i^j, u_j, v_j) : x_i \in d_j\} \cup \{(\overline{x}_i^j, u_j, v_j) : \overline{x}_i \in d_j\}$.

Положим $D = \cup_{i=1,\ldots,m} D_i$. Очевидно $|D| = 3m$.

Определим множества

$$G = \{g_1, g_2, \ldots, g_k\},\ |G| = m(n-1),$$
$$H = \{h_1, h_2, \ldots, h_k\},\ |H| = m(n-1),$$

где $k = m(n-1)$. Определим также множество троек

$$P = \{(x_i^j, g_l, h_l), (\overline{x}_i^j, g_l, h_l), i \in \overline{1,n}, j \in \overline{1,m}, l \in \overline{1,k}\},$$

$|P| = 2n\,m\,m(n-1) = 2m^2n(n-1)$.

Теперь сформулируем задачу о трехмерном сочетании. Положим

$$X = \cup_{i \in \overline{1,n}} X_i,$$
$$Y = \cup_{i \in \overline{1,n}} A_i \cup \cup_{j=\overline{1,m}} u_j \cup \cup_{l=\overline{1,k}} g_l$$
$$Z = \cup_{i \in \overline{1,n}} B_i \cup \cup_{j=\overline{1,m}} v_j \cup \cup_{l=\overline{1,k}} h_l$$

Очевидно, что $|X| = |Y| = |Z| = 2mn$. Положим $q = 2mn$. Определим множество $M = T \cup D \cup P$. Сформулированная задача имеет решение, если в данном множестве найдется 3-сочетание из q троек.

Пусть такое 3-сочетание R нашлось. Рассмотрим множество $R_P = R \cap P$. Так как число различных элементов в последней позиции не более $m(n-1)$, то, очевидно,$|R_P| \leq m(n-1)$. Аналогично показывается, что множество $R_D = R \cap D$ содержит не более m троек.

Рассмотрим множество $R_T^i = R \cap T_i$. Из определения множества T_i следует, что оно содержит не более m элементов, так как если в нем есть тройка содержащая x_i^j, то в него не входит никакая тройка, содержащая $\overline{x}_i^j$. Таким образом, $|R_T^i| \leq m$, а общее число элементов в R_T не превосходит mn. Суммируя сказанное, получим, что с одном стороны $|R| = |R_T| + |R_D| + |R_P| \leq mn + m + m(n-1) = 2mn$. С другой стороны по определению 3-сочетания $|R| = 2mn$. Значит $|R_T| = mn$, $|R_T^i| = m, i = 1, \ldots, n$, $|R_D| = m$, $|R_P| = m(n-1)$.

Рассмотрим множество R_T^i. Так как $|R_T^i| = m$, то в это множество входят либо все тройки множества T_i^t, либо все тройки множества T_i^f. Действительно, любая тройка $(\overline{x}_i^j, a_i^j, b_i^j)$ из T_i^t имеет пересечения по компонентам с еще двумя тройками $(x_i^j, a_i^{j'}, b_i^j)$ и $(x_i^{j''}, a_i^j, b_i^{j''})$, из T_i^f. Через j' и j'' обозначены индексы, вычисляемые по формулам:

$$j' = \begin{cases} j+1, \text{ если } j < m, \\ 1, \text{ если } j = m. \end{cases}$$

$$j'' = \begin{cases} j-1, \text{ если } j > 1, \\ m, \text{ если } j = 1. \end{cases}$$

Аналогично показывается, что любая тройка $(x_i^j, a_i^{j'}, b_i^j)$ из T_i^f имеет общие компоненты с ровно двумя тройками $(\overline{x}_i^j, a_i^j, b_i^j)$, $(\overline{x}_i^{j'}, a_i^{j'}, b_i^{j'})$ из T_i^t. Таким образом, наличие в 3-сочетании любой тройки запрещает нахождение там еще двум тройкам (из другого семейства). И каждая тройка не сочетается в точности с двумя тройками (из другого семейства). Из этого следует, что m троек в сочетании возможны только в случае, если это тройки из одного из семейств T_i^t либо T_i^f. Таким образом, возможно два случая: $R_T^i = T_i^t$ либо $R_T^i = T_i^f$.

Заметим, что данная задача сводится к следующей: пусть имеется $2k$ точек на окружности. Выбрать подмножество точек максимальной мощности, не содержащее двух соседних точек. Очевидно, что данное подмножество имеет мощность k и состоит из точек, идущих через одну. Точкам ставятся в соответствие тройки, тройки содержащие переменную и ее отрицание, чередуются.

Пусть $R_D^j = R \cap D_j$. В силу равенства последних двух компонентов всех троек множества D_j во множество R_D^j не может входить более одной тройки из D_j. Как уже было отмечено $|R_D| = m$. Следовательно

$|R_D^j| = 1$, т.е. множество R_D^j содержит в точности одну тройку. Рассмотрим случай $R_T^i = T_i^t$. Так как в первой позиции у них стоит $\overline{x}_i^j, j \in \overline{1,m}$, то тройка из R_D^j не может иметь в первой позиции $\overline{x}_i^j$. Это означает, что в в эту тройку переменная x_i входит без отрицания или не входит. Аналогично показывается, что во втором случае, т.е. когда все тройки лежат в множестве T_i^f переменная x_i, напротив входит в тройку под знаком отрицания или не входит. Из этого следует, что если для $i = 1, \ldots, n$ присвоить переменной x_i значение 1 в случае, когда $R_T^i = T_i^t$ и 0, если $R_T^i = T_i^f$, то получим выполняющий набор.

Аналогично показывается (упражнение 6), что если есуществует выполняющий набор для булевой формулы в задаче выполнимости, то соответствующая задача о трехмерном сочетании разрешима. Таким образом, построенная задача о трехмерном сочетании разрешима тогда и только тогда, когда разрешима рассматриваемая задача выполнимости.

Полиномиальность сводимости также очевидна. Тем самым утверждение доказано. □

Упражнение 6 Завершите доказательство утверждения 3, установив, что если существует выполняющий набор для булевой формулы в задаче выполнимости, то соответствующая задача о трехмерном сочетании разрешима.

1.5.3 Задача о сумме подмножеств в форме распознавания

Задача о сумме подмножеств может быть сформулирована в двух постановках: в форме максимизации и в форме распознавания. Первый вариант будет рассмотрен в дальнейших главах. Рассмотрим постановку задачи в форме распознавния.

Дан набор n натуральных чисел $w_1, \ldots, w_n$ и натуральное число C. Требуется определить, существует ли такое подмножество I множества $\overline{1,n}$, что $\sum_{i \in I} w_i = C$.

Утверждение 4 Задача о сумме подмножеств в форме распознавания является NP-полной.

Доказательство. Для доказательства достаточно показать, что задача о трехмерном сочетании может быть полиномиально сведена к эквивалентной задаче о сумме подмножеств. Рассмотрим задачу о трехмерном

сочетании, в которой заданы три множества X, Y, Z мощности q каждое:

$$X = \{x_1, \ldots, x_q\},$$
$$Y = \{y_1, \ldots, y_q\},$$
$$Z = \{z_1, \ldots, z_q\}$$

и множество $M \subseteq X \times Y \times Z$ мощностью k:

$$M = \{m_1, \ldots, m_k\}.$$

Требуется определить, существует ли подмножество $M' \subseteq M$, являющееся 3-сочетанием. Произвольный элемент m_i множества M представляет собой тройку $(x_{f(i)}, y_{g(i)}, z_{h(i)})$, $x_{f(i)} \in X, y_{g(i)} \in Y, z_{h(i)} \in Z$. Заметим, что индексы $f(i), g(i), h(i)$ лежат в интервале $\overline{1,q}$. Сопоставим числу $t \in \overline{1,q}$ двоичный код $\sigma(t)$ из pq компонент, состоящий из q групп по p двоичных позиций, где $p = \lceil \log_2(k+1) \rceil$. При этом все разряды этого двоичного кода содержат 0, кроме единственной единичной компоненты в позиции $p(t-1)+1$. Таким образом каждому числу $t \in \overline{1,q}$ сопоставляется 1 в первой позиции группы с номером t. Каждой тройке $m_i = (x_{f(i)}, y_{g(i)}, z_{h(i)})$ из M сопоставим двочичное число $s(m_i) = \sigma(x_{f(i)})\sigma(y_{g(i)})\sigma(z_{h(i)})$. Таким образом код $s(m_i)$ содержит $3q$ групп по p двоичных позиций. Каждая группа соответствет одному из элементов множества $X \cup Y \cup Z$ и содержит двоичное представление 1, если соответствующий элемент входит в m_i и 0 — в противном случае.

Так как $p = \lceil \log_2 k \rceil + 1$, то при суммировании чисел $s(m_i)$ по любому подмножеству M' множества M не будет производиться перенос разрядов между группами из двоичных позиций. Каждая такая группа будет содержать двоичное представление числа, равного количеству вхождений соответствующего элемента в тройки из M'. Подмножество M' будет 3-сочетанием тогда и только тогда, когда двоичное представление числа, равного сумме $\sum_{m \in M'} s(m)$ содержит 1 в первой позиции каждой группы из p разрядов. Определим множество $W = \{w : s(m) = w, m \in M\}$. Задача о трехмерном сочетании имеет решение тогда и только тогда, когда найдется такое подмножество W' множества W, что

$$\sum_{w \in W'} w = V, \text{ где } V = \sum_{i=0}^{3q-1} 2^{pi}.$$

Число V и каждое из чисел в множестве W имеет двоичное представление длиной не более $3pq$. Поэтому формальная запись условий задачи о сумме подмножеств может быть сгенерирована не более чем за полиномиальное время от длины записи задачи о трехмерном сочетании.

Тем самым установлена полиномиальная сводимость задачи о трехмерном сочетании к задаче о сумме подмножеств и, как следствие, показана NP-полнота задачи о сумме подмножеств.

□

Пример 2 Составить и решить задачу о сумме подмножеств для задачи о 3*-сочетании:*

$$X = x_1, x_2, x_3,$$
$$Y = y_1, y_2, y_3,$$
$$Z = z_1, z_2, z_3.$$

$$M = \{\{x_1, y_2, z_1\},$$
$$\{x_2, y_1, z_2\},$$
$$\{x_1, y_2, z_2\},$$
$$\{x_3, y_3, z_3\}\}.$$

1.6 NP-полнота в сильном смысле

Рассмотрим задачу о сумме подмножеств в форме распознавания. Дан набор n натуральных чисел $w_1, \ldots, w_n$ и натуральное число C. Требуется определить, существует ли такое подмножество I множества $\overline{1,n}$, что $\sum_{i \in I} w_i = C$.

Данная задача может быть решена методом динамического программирования. Рассмотрим задачу о сумме подмножеств $P(m, B)$, в которой фигурируют только m переменных $x_1, \ldots, x_m$ с правой частью B. Рассмотрим функцию $f(m, j)$ двух переменных, принимающую значение T, если задача $P(m, B)$ имеет ршение и F в противном случае. Выпишем уравнение Беллмана при $m > 1, B > 0$:

$$f(m, B) = \begin{cases} T \text{ если } f(m-1, B) = T \text{ или } f(m-1, B - w_m) = T \\ F \text{ в противном случае.} \end{cases}$$

Считаем

$$f(m, B) = T \text{ при } B = 0,$$
$$f(m, B) = F \text{ при } B < 0.$$

Процесс решения можно представить в виде таблицы с n строками и $C + 1$ столбцами. Крайний левый столбец заполняется нулями. Выполняется последовательное заполнение строк таблицы, начиная с первой.

Задача считается решенной, как только в крайнем правом стобце появляется значение T. Либо после заполнения всей таблицы крайний правый столбец содержит только значения F.

Упражнение 7 Решить задачи о сумме подмножеств в форме распознавания

$$5x_1 + 6x_2 + 3x_3 + 4x_4 = 7,$$
$$5x_1 + 6x_2 + 3x_3 + 4x_4 = 9.$$

Максимальное число шагов, нужное для решения задачи данным методом, не превосходит величины nC. Объем входных данных задачи примерно равен $N = \sum_{i=1}^{n} \log_2(w_i) + \log_2 C$. Если величина коэффициентов ограничена константой $w_i \leq w$, $i = 1, \ldots, n$, то длина исходных данных лежит в интервале $[n, n \log w]$, а сложность в худшем случае является полиномом второй степени от w, т.к. $C \leq \sum_{i=1}^{n} w_i \leq nw$.

Для задач с числовыми параметрами вводится понятие *псевдополиномиального алгоритма*, т.е. алгоритма, сложность которого ограничена полиномом от двух величин n — длина входа и m —- величина максимального числа.

Задача называется NP-полной в сильном смысле, если для ее решения не существует псевдополиномиального алгоритма.

Глава 2

Задача о ранце

2.1 Постановка и базовые свойства

Задача о ранце [1],[2] с одним ограничением является одной из классических задач дискретной оптимизации, применяющейся при моделировании различных экономических процессов, решении проблем, возникающих в промышленном производстве, планировании, управлении и других сферах. Различным вопросам, связанным с данной задачей, посвящено большое число исследований, статей и монографий.

Задача о ранце формулируется следующим образом. Даны n предметов. Предмет i характеризуется весом w_i и ценой p_i. Требуется положить в ранец грузоподъемностью C набор предметов максимальной стоимости. Данное неформальное описание может быть математически записано следующим образом:

$$\begin{aligned} &f(\overline{x}) = \textstyle\sum_{i=1}^{n} p_i x_i \to \max; \\ &\textstyle\sum_{i=1}^{n} w_i x_i \le C; \\ &x_i \in \{0,1\}, i = 1, \ldots, n. \end{aligned} \tag{2.1}$$

При этом обычно предполагаются справедливыми неравенства $w_i \le C$, $\sum_{i=1}^{n} w_i > C$. Функция $f(\overline{x})$, где через $\overline{x}$ обозначается набор $(x_1, \ldots, x_n)$, называется *целевой функцией* для данной задачи.

Задачу о ранце можно решать путем полного перебора всех кортежей длиной n. При этом для решения задачи размерности n в наихудшем случае потребуется просмотреть 2^n кортежей. Сделать поиск решения более эффективным позволяет метод ветвей и границ. Этот метод заключается в последовательной декомпозиции исходной задачи на подзадачи, с от-

севом подзадач, решение которых заведомо не приведет к нахождению оптимума исходной задачи.

Дадим общее описание метода ветвей и границ [1]–[3] для решения задачи о ранце.

Для удобства описания метода ветвей и границ введем в рассмотрение более общую постановку задачи о ранце, в которой предполагается, что часть переменных принимают фиксированные значения:

$$\begin{aligned} &\textstyle\sum_{i=1}^{n} p_i x_i \to max; \\ &\textstyle\sum_{i=1}^{n} w_i x_i \le C; \\ &x_i = \theta_i, i \in I, \theta_i \in \{0,1\} \\ &x_i \in \{0,1\}, i \in N \setminus I, \end{aligned} \tag{2.2}$$

где I — множество индексов всех переменных x_i, принимающих фиксированные значения θ_i. Введенная постановка позволяет описывать подзадачи, возникающие в процессе работы метода ветвей и границ. Отметим, что для задача (2.1) является частным случаем задачи (2.2), когда множество I пусто.

Задаче (2.2) соответствует следующая линейная **задача релаксации**:

$$\begin{aligned} &\textstyle\sum_{i=1}^{n} p_i x_i \to max; \\ &\textstyle\sum_{i=1}^{n} w_i x_i \le C; \\ &x_i = \theta_i, i \in I, \theta_i \in \{0,1\} \\ &0 \le x_i \le 1, i \in N \setminus I. \end{aligned} \tag{2.3}$$

Оптимум задачи (2.3) не меньше оптимума задачи (2.2). Поэтому, если оптимальное решение задачи (2.3) достигается на целочисленном наборе значений $x_1, \ldots, x_n$, то этот набор является также оптимальным решением задачи (2.2).

Задача релаксации решается за линейное относительно числа переменных количество операций методом Данцига [1], [2]. Известно, что ее оптимальное решение достигается на наборе значений переменных $x_1^r, \ldots, x_n^r$,

содержащем не более одного дробного (не целого) значения:

$$\begin{array}{l} x_i^r = 1, 1 \le i \le s-1, i \in N \setminus I, \\ x_s^r = \left(C - \sum_{i \in I} \theta_i w_i - \sum_{i \in \{1,\ldots,s-1\} \setminus I} w_i\right) / w_s, \\ x_i^r = 0, s+1 \le i \le N, i \in N \setminus I, \\ x_i^r = \theta_i, i \in I. \end{array} \tag{2.4}$$

где индекс s **дробной переменной** x_s^r определяется по формуле

$$s = \min i : \sum_{j \in \{1,\ldots,i\} \setminus I} w_j > C - \sum_{j \in I} \theta_j w_j, \ i \in N \setminus I.$$

Оценка, получаемая с использованием релаксационной задачи, вычисляется по следующей формуле:

$$U_{LR} = \left\lfloor \sum_{i \in \{1,\ldots,s\} \setminus I} p_i + \sum_{i \in I} \theta_i p_i + \frac{p_s}{w_s} \left(C - \sum_{i \in I} \theta_i w_i - \sum_{i \in \{1,\ldots,s-1\} \setminus I} w_i \right) \right\rfloor . \tag{2.5}$$

Если же $\sum_{i \in N \setminus I} w_i \le C - \sum_{i \in I} \theta_i w_i$ и искомого s не существует, то решением задачи релаксации будет единичный набор, а $U_{LR} = \sum_{i \in N \setminus I} p_i + \sum_{i \in I} \theta_i p_i$.

В процессе работы алгоритма поддерживаются следующие данные: **рекорд**, т.е. наибольшее найденное на данный момент времени значение целевой функции f, **рекордное решение**, на котором достигается рекорд, и текущий список *подзадач*, на которые разбита исходная задача. Рекорд выполняет роль нижней оценки оптимума. Подзадачи из списка являются задачами вида (2.2).

Данные: рекорд f^0, рекордное решение x^0, список подзадач.

Шаг 1. В список подзадач помещается исходная задача. Рекорд полагается равным 0.

Шаг 2. Если список подзадач пуст, то алгоритм завершается. В противном случае выбирается подзадача P из списка подзадач. Подзадача P удаляется из списка.

Шаг 3. Проверяется, выполнены ли для выбранной подзадачи P указанные далее в работе *условия отсева*. Если одно из условий отсева выполнено, то осуществляется переход к шагу 2. При необходимости на этом шаге также обновляются рекорд и соответствующее рекордное решение.

Шаг 4. Выбранная подзадача подвергается декомпозиции. Для этого выбирается переменная x_b, называемая *переменной ветвления*. Подзадача P вида (2.2) разбивается на две подзадачи P_0 и P_1, получаемые

присваиванием переменной x_b значений 0 и 1 соответственно:
подзадача P_0:

$$\sum_{i=1}^{n} p_i x_i \to max;$$

$$\sum_{i=1}^{n} w_i x_i \leq C;$$

$$x_i = \theta_i, i \in I, x_b = 0,$$

$$x_i \in \{0, 1\}, i \in N \setminus \{I \cup b\},$$

подзадача P_1:

$$\sum_{i=1}^{n} p_i x_i \to max;$$

$$\sum_{i=1}^{n} w_i x_i \leq C;$$

$$x_i = \theta_i, i \in I, x_b = 1,$$

$$x_i \in \{0, 1\}, i \in N \setminus \{I \cup b\},$$

Построенные подзадачи P_0 и P_1 помещаются в список подзадач и осуществляется переход к шагу 2.

На шаге 3 метода ветвей и границ вычисляются верхние и нижние оценки. Нижней оценкой является наибольшее найденное к данному шагу значение целевой функции (рекорд). Вычисление рекорда f_r проводится следующим образом. Перед началом расчетов полагаем значение рекорда равным нулю $f_r = 0$, которое достигается на нулевом рекордном решении $x_r = 0$. Если $\sum_{i \in I} \theta_i w_i + \sum_{j \in N \setminus I} w_i \leq C$, то вектор $\hat{x}$, где

$$\hat{x}_i = \begin{cases} \theta_i, i \in I, \\ 1, i \in N \setminus I, \end{cases}$$

будет допустимым решением задачи (2.6). Если $f(\hat{x}) \geq f_r$ то производится обновление рекорда $f_r := f(\hat{x}), x_r := \hat{x}$.

Декомпозиция, выполняемая на шаге 4 МВГ, состоит в разбиении исходной задачи на две путем присваивания одной из переменных значений 0 и 1 соответственно и называется *ветвлением* задачи по переменной. Наиболее распространенными способами выбора переменной для ветвления являются выбор первой переменной в соответствии с некоторым порядком или выбор дробной переменной в задаче релаксации. Пример первого подхода можно найти в работе [4], а второго — в работе [5]. Ветвление по дробной переменной считается стандартным и предлагается в качестве основного в некоторых учебных курсах [3].

Результатом работы алгоритма является окончательное рекордное решение. Заметим, что работа описанного нами алгоритма существенным образом зависит от процедуры выбора очередной подзадачи из списка подзадач и процедуры выбора переменной ветвления для декомпозиции выбранной подзадачи. Алгоритм, для которого данные процедуры строго определены, будем называть *вариантом* метода ветвей и границ.

Исключение подзадач из дальнейшего рассмотрения (отсев) производится на шаге 3 с помощью специальных *правил отсева*. В стандартном варианте метода ветвей и границ для задачи о ранце решается так называемая *оценочная задача*, которая позволяет получить верхнюю оценку для решения рассматриваемой подзадачи. В качестве оценочной часто выбирают линейную релаксацию задачи (2.3). Вариант МВГ будем называть *стандартным*, если условиями отсева выбранной подзадачи P вида (2.2) являются следующие условия:

1. **C1**: $\sum_{i\in I}\theta_i w_i > C$, т.е. подзадача P не имеет решения;
2. **C2**: Оценка U_{LR}, полученная с помощью задачи (2.3) не превосходит значение текущего рекорда, тем самым оптимальное значение целевой функции f для подзадачи P также заведомо не превосходит значение текущего рекорда.

Вариант МВГ будем называть *ослабленным*, если условиями отсева выбранной подзадачи P вида (2.2) являются условие **C1** и условие **C2′**: $\sum_{i\in I}\theta_i w_i + \sum_{i\in N\setminus I} w_i \le C$, т.е. оптимальным решением подзадачи P является набор значений $x_1,\ldots,x_n$, такой что $x_i=\theta_i$ при $i\in I$ и $x_i=1$ при $i\in N\setminus I$.

Утверждение 5 Если для некоторой подзадачи задачи (2.1) выполнено условие отсева **C2′**, *то условие C2 также выполнено для этой подзадачи.*

Доказательство. Пусть для подзадачи P выполнено условие отсева **C2′**. Тогда $\sum_{i\in I}\theta_i w_i + \sum_{i\in N\setminus I} w_i \le C$. Из этого следует, что решение задачи x' релаксации для подзадачи P является целочисленным и совпадает с решением подзадачи P. Согласно правилу обновления рекорда, $f(x')\le f_r$, поэтому данная подзадача может быть исключена из дальнейшего рассмотрения по правилу отсева **C2**. □

Процесс работы метода ветвей и границ можно схематично представить в виде *дерева ветвлений* — ациклического ориентированного графа, вершинами которого являются обрабатываемые подмножества, а дуги

соединяют одно подмножество с другими, полученными из него в результате декомпозиции, выполненной на шаге 3 алгоритма.

Сложностью метода ветвей и границ при решении данной задачи будем называть число итераций цикла 2-4. Очевидно, что это число совпадает с количеством вершин в дереве ветвлений на момент завершения работы алгоритма. Известно, что число вершин дерева S связано с числом его концевых вершин соотношением $S = 2 * V - 1$. Количество концевых вершин дерева ветвлений будем называть *приведенной сложностью.*

2.2 Задача о сумме подмножеств

Частным случаем задачи о ранце является задача о сумме подмножеств (subset-sum problem), в которой веса и цены совпадают:

$$\begin{array}{l} f(x) = \sum_{i=1}^{n} w_i x_i \to \max, \\ g(x) = \sum_{i=1}^{n} w_i x_i \le C, \\ x_i \in \{0,1\}, i \in \overline{1,n}. \end{array} \tag{2.6}$$

Приведенную постановку будет называть *задачей о сумме подмножеств в оптимизационноной форме*. Рассмотрим также задачу о сумме подмножеств *в форме распознавания*:

$$\begin{array}{l} \text{Существует ли такой набор } \{x_i\}, \text{ что} \\ \sum_{i=1}^{n} w_i x_i = C, \\ x_i \in \{0,1\}, i \in \overline{1,n}. \end{array} \tag{2.7}$$

2.2.1 Методы ветвей и границ

Сформулируем правила отсева. Заметим, что для задачи о сумме подмножеств правила **C2′** и **C2** эквивалентны. Для этого достаточно показать справедливость следующего утверждения.

Утверждение 6 Если для некоторой подзадачи задачи (2.6) не выполнено условие отсева **C2′**, *то условие* **C2** *также не выполнено для этой подзадачи.*

Доказательство. Пусть для подзадачи P не выполнено условие отсева **C2′**, т.е. $\sum_{i \in I} \theta_i w_i + \sum_{i \in N \setminus I} w_i > C$. Из этого следует, что оценка релаксации составляет $U_{LR} = C$. Поскольку значение рекорда не может быть больше, чем C, данная оценка бесполезна в правилах отсева. □

В работе [6] приводится пример следующей задачи:

$$f(\overline{x}) = \sum_{i=1}^{n} 2x_i \to max; \sum_{i=1}^{n} 2x_i \leq 2\left\lfloor \frac{n}{2} \right\rfloor + 1; x_i \in \{0,1\}, i = 1, \ldots, n, \quad (2.8)$$

и показывается, что сложность решения этой задачи методом ветвей и границ при любом способе выбора очередной подзадачи и переменной для ветвления составляет

$$\Phi(n) = 2\binom{n+1}{\lfloor \frac{n}{2} \rfloor + 1} - 1. \quad (2.9)$$

Подзадачи, отсеянные по правилу **C1**, соответствуют наборам зафиксированных переменных, которые содержат в точности $\lfloor \frac{n}{2} \rfloor + 1$ единиц. Таких вершин столько, сколько может быть сочетаний из n по $\lfloor \frac{n}{2} \rfloor + 1$, т.е. $\binom{n}{\lfloor \frac{n}{2} \rfloor + 1}$. Подзадачи, отсеянные по правилу **C2′** соответствуют наборам, в которых зафиксировано $n - \lfloor \frac{n}{2} \rfloor$ нулей. Таких наборов будет $\binom{n}{n - \lfloor \frac{n}{2} \rfloor} = \binom{n}{\lfloor \frac{n}{2} \rfloor}$. Получаем, что общее число концевых вершин составляет $\binom{n}{\lfloor \frac{n}{2} \rfloor + 1} + \binom{n}{\lfloor \frac{n}{2} \rfloor} = \binom{n+1}{\lfloor \frac{n}{2} \rfloor + 1}$. Отсюда получаем общее число вершин в дереве ветвлений, вычисляемое по формуле 2.9. □

Заметим, что выбор ценовых и весовых коэффициентов равными 2 в целевой функции задачи (2.8) на первый взгляд может показаться излишним усложнением. Более естественным представляется положить их равными 1. Однако, в таком случае, верхняя оценка $U_{LR} = \lfloor \lfloor \frac{n}{2} \rfloor + \frac{1}{2} \rfloor = \lfloor \frac{n}{2} \rfloor$ очевидно совпадает с рекордом и задача решается как только этот рекорд будет найден.

Возникает вопрос о точности данной оценки. В работе [7] доказывается, что если для ветвления выбирается каждый раз переменная, соответствующая предмету с максимальным весом, то (2.9) будет точной верхней оценкой для сложности такого варианта МВГ. В работе [8] показано, что если для ветвления выбирается дробная переменная, то можно построить примеры, в которых сложность базового варианта метода ветвей и границ превзойдет $\Phi(n)$.

Сложность решения задачи о сумме подмножеств методов ветвей и границ во-многом определяется коэффициентами задачи. В работе доказывается справедливость следующих верхних оценок сложности.

Теорема 3 Сложность S решения задачи (2.6) методом ветвей и границ при любом порядке выбора подзадач для обработки и переменных для ветвления удовлетворяет следующему неравенству:

$$S \leq 2\binom{n+1-s+t}{t} - 1, \quad (2.10)$$

где величины s и t определяются соотношениями:

$$s = \min\left\{k \in N : \sum_{i=1}^{k} w_i > C\right\}. \tag{2.11}$$

$$t = \min\left\{k \in N : \sum_{i=n-k+1}^{n} w_i > C\right\}. \tag{2.12}$$

Величины (2.11) и (2.12) определены в предположении, что веса упорядочены по убыванию $w_1 \geq \cdots \geq w_n$.

2.2.2 Методы динамического программирования

Методы динамического программирования основаны на принципе оптимальности Беллмана. Для задачи о сумме подмножеств данный принцип может быть сформулирован следующим образом:

$$f^*(m, c) = \max(f^*(m-1, c), w_m + f^*(m-1, c-w_m)),$$
$$f^*(m, c) = 0 \text{ если } m = 0 \text{ или если } c \leq 0.$$

где $f^*(m, C)$ при $1 \leq m \leq n$ и $1 \leq c \leq C$ — оптимальное решение подзадачи о сумме подмножеств вида

$$\begin{array}{l} f(x) = \sum_{i=1}^{m} w_i x_i \to \max, \\ g(x) = \sum_{i=1}^{m} w_i x_i \leq C, \\ x_i \in \{0, 1\}, i \in \overline{1, m}. \end{array} \tag{2.13}$$

В базовом табличном варианте метода динамического программирования происходит построчное заполнение таблицы сверху вниз. Строкам соответствуют переменные, а столбцам — значения правой части ограничения задачи .

m \ c	0	1	...	C
0	0	0	...	0
1	0	$f^*(1,1)$	...	$f^*(1,C)$
		...		
		...		
n	0	$f^*(n,1)$	...	$f^*(n,C)$

В результате после заполнения таблицы элемент $f^*(n, C)$ будет содержать оптимальное значение целевой функции в задаче (2.6). Заметим, что процесс расчетов можно остановить ранее, если в правой ячейке i-го ряда получено значение $f^*(i, C) = C$. Число шагов алгоритма в случае,

если расчеты проводятся до конца, пропорционально nC, также как и затраты памяти. Широко известен базовый табличный вариант динамического программирования для задачи о ранце. В этом варианте строится таблица, в которой строки соответствуют переменным, а столбцы — возможным целочисленным значениям целевой функции. Добавляются еще вспомогательные столбец для нулевого значения целевой функции $c = 0$, и строка для нулевой переменной $i = 0$, не имеющие содержательного смысла, но используемые в вычислениях. Таким образом, в таблице $n+1$ строк и $C+1$ столбов. На первой фазе табличный алгоритм динамического программирования (TabDP) последовательно заполняет строки таблицы в направлении слева-направо:

цикл $c = 0$ **до** C **выполнять**
| $g(0, c) := 0$
конец
цикл $i = 0$ **до** n **выполнять**
| $g(i, 0) := 0$
конец
цикл $i = 1$ **до** n **выполнять**
| **цикл** $c = 0$ **до** C **выполнять**
| | **Если** $c < w_i$ **, тогда**
| | | $g(i, c) := g(i-1, c)$
| | **иначе**
| | | $g(i, c) := \max(g(i-1, c), g(i-1, c-w_i) + w_i)$
| | **Конец условия**
| **конец**
конец

Алгоритм 1: TabDP

Алгоритм завершается, как только на очередном шаге i^* значение $g(i^*, C)$ становится равным C, либо $i^* = n$. Оптимальное значение целевой функции находится в элементе таблицы $g(i^*, C)$. Вторая фаза алгоритма восстанавливает вектор x по таблице g.

Пример 3 Пример.

$$4x_1 + 2x_2 + 4x_3 \to \max$$
$$4x_1 + 2x_2 + 4x_3 \leq 5.$$

Решение:

m \ c	*0*	*1*	*2*	*3*	*4*	*5*
0	*0*	*0*	*0*	*0*	*0*	*0*
1	*0*	*0*	*0*	*0*	*4*	*4*
2	*0*	*0*	*2*	*2*	*4*	*4*
3	*0*	*0*	*2*	*2*	*4*	*4*

Для задачи о сумме подмножеств существует более эффективная реализация метода динамического программирования, в которой используется только один вектор длиной C. В ячейке вектора с номером $\hat{c}$ хранится 0, если на данном шаге не найдено такое решение x, что $f(x) = \hat{c}$ и номер переменной, соответствующей последнему добавленному весу в противном случае.

Для примера 3 последовательность шагов будет выглядеть следующим образом:

m \ c	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	0
2	0	0	2	0	1	0
3	0	0	2	0	1	0

Максимум целевой функции соответствует номеру максимальной ненулевой ячейки в полученном векторе. Для восстановления решения на обратном ходу

2.2.3 Субэкспоненциальный алгоритм

Рассмотрим следующую задачу. Пусть даны две последовательности чисел $a_1, \ldots, a_k$ и $b_1, \ldots, b_k$. Требуется определить, существует ли такая пара чисел a_i, b_j, что $a_i + b_j = C$. Тривиальный алгоритм проверки всех возможных пар имеет сложность k^2. Более интеллектуальный алгоритм состоит следующем. На первой фазе выполняется предварительная сортировка элементов обеих последовательностей, что имеет сложность $O(k \log_2 k)$. Затем выполняется перебор всех элементов первой последовательности. Для каждого выбранного элемента a_i во второй последовательности выполняется поиск элемента $b_j = C - a_i$. При условии, что элементы второго массива отсортированы, такой поиск может быть выполнен за $O(\log_2 k)$ операций. Таким образом, рассмотренная задача может быть решена за $O(k \log_2 k)$ операций.

Применим данный подход к решению задачи о сумме подмножеств в форме распознавания. Напомним ее формулировку. Дан набор n нату-

ральных чисел $w_1, \ldots, w_n$ и натуральное число C. Требуется определить, существует ли такое подмножество I множества $\overline{1,n}$, что $\sum_{i \in I} w_i = C$. Пусть $A = \{a \in \mathbb{N} : a = \sum_{i \in I} w_i, I \subseteq 1, \ldots, \lfloor n/2 \rfloor\}$, $B = \{b \in \mathbb{N} : b = \sum_{i \in I} w_i, I \subseteq \lfloor n/2 \rfloor + 1, \ldots, n\}$. Будем считать, что I может быть пустым множеством, т.е. $0 \in A$ и $0 \in B$.

Множества A и B могут быть получены за $O(2^{n/2})$ операций простым перебором. Далее, воспользовавшись решением рассмотренной ранее задачи, за $O(\frac{n}{2}2^{n/2})$ операций можно получить ответ на вопрос о том, существует ли такая пара чисел a, b, $a \in A$ и $b \in B$, что $a + b$ и тем самым решить задачу о сумме подмножеств. Таким образом, получаем, что задача о сумме подмножеств может быть решена за $O(n2^{n/2})$ операций.

Для решения задачи о сумме подмножеств в форме задачи оптимизации (2.6) нужно несколько модифицировать вспомогательную задачу. Пусть даны две последовательности чисел $a_1, \ldots, a_k$ и $b_1, \ldots, b_k$. Требуется определить максимальное значение $z = a_i + b_j$ при условии $z \leq C$. На первой фазе выполняется предварительная сортировка элементов обеих последовательностей, что имеет сложность $O(k \log_2 k)$. Затем выполняется перебор всех элементов первой последовательности. Для каждого выбранного элемента a_i во второй последовательности выполняется поиск максимального элемента $b_j \leq C - a_i$. При условии, что элементы второго массива отсортированы, такой поиск может быть выполнен за $O(\log_2 k)$ операций путем последовательного разбиения списка пополам. Таким образом, рассмотренная задача может быть решена за $O(k \log_2 k)$ операций.

Теперь рассмотрим задачу о ранце вида (2.1). Для ее решения рассмотрим следующую вспомогательную задачу. Пусть даны четыре последовательности натуральных чисел $a_1, \ldots, a_k, b_1, \ldots, b_k, c_1, \ldots, c_k$ и $d_1, \ldots, d_k$. Нужно найти такую пару чисел a_i, b_j, что $a_i + b_j \leq C$ и $c_i + d_j$ при этом максимально. Тривиальное решение, сводящееся к полному перебору, требует k^2 операций. Заметим, что если для пар (a_i, c_i) и (a_j, c_j) выполнены соотношения $a_i \leq a_j$ и $c_i \geq c_j$, то пара (a_j, c_j) может не рассматриваться. Аналогичное утверждение справедливо в отношении пар (b_i, d_i) и (b_j, d_j), таких что $b_i \leq b_j$ и $d_i \geq d_j$. Предположим, что массивы $a_1, \ldots, a_k$ и $c_1, \ldots, c_k$ образуют только недоминируемые пары и упорядочены по возрастанию. Аналогичное предположение сделаем в отношении массивов $b_1, \ldots, b_k$, $d_1, \ldots, d_k$. Тогда, выбирается одно из чисел $b_1, \ldots, b_k$ а затем методом деления пополам за $O(log_2 k)$ находится такое число a_i, что $a_i + b_j \leq C$. Очевидно, что при условии недоминируемости это обеспечивает оптимальность полученного решения. Учитывая, что сортировка с отсевом доминируемых пар может быть выполнена за $O(k \log_2 k)$ операций, получаем, что данная задача может быть решена за $O(k \log_2 k)$.

2.2.4 Концепция ядра и баланисировки в динамическом программировании

Здесь и далее будем предполагать, что предметы упордочены в порядке невозрастания эффективности: $p_1/w_1 \geq \cdots \geq p_n/w_n$. Будем называть *жадным* решение x^G, определяемое соотношениями:

$$x_i^G = \begin{cases} 1, i \in \overline{1, s-1}, \\ 0, i \in \overline{s, n}, \end{cases}$$

$s = \min\left\{k \in N : \sum_{i=1}^{k} w_i > C\right\}$.

Известно [1],[2] что как правило оптимальное решение задачи о ранце отличается от "жадного" в некотором наборе переменных, индексы которых отличаются от s не более чем на некоторую величину d, много меньшую n. Множество индексов, в которых некоторое решение отличается от "жадного", называется его *ядром*. Для набора x введем обозначения для границ ядра:

$$a(x) = \min_{1 \leq i \leq s-1} \{x_i = 0\}$$
$$b(x) = \max_{s \leq i \leq n} \{x_i = 1\}$$

Весом набора x будем называть величину $w(x) = \sum_{i=1}^{n} x_i w_i$, а его стоимостью — величину $p(x) = \sum_{i=1}^{n} x_i w_i$. Набор x называется *сбалансированным*, если $|w(x) - C| \leq w_{max}$. Легко заметить, что для оптимального решения задачи о ранце (2.1) суммарный вес входящих в него предметов, отличается от C не более чем на максимальный вес предмета $w_{max} = \max_{i=1,\ldots,n} w_i$. Таким образом, оптимальное решение является сбалансированным набором.

Введем в рассмотрение следующие операции над сбалансированным набором x.

1. *Вставка*: если $\sum_{i=1}^{n} x_i w_i \leq C$ и $b(x) < n$, то положить $x_i = 1$, где $b(x) < i \leq n$.

2. *Удаление*: если $\sum_{i=1}^{n} x_i w_i > C$ и $a(x) > 1$, то положить $x_i = 0$, где $1 \leq i < a(x)$.

Легко показать справедливость следующего утверждения.

Утверждение 7 В результате применения к сбалансированному набору одной из операций Вставка или Удаление будет получен сбалансированный набор.

Менее тривиальным является следующее утверждение.

Утверждение 8 Любое оптимальное решение задачи (2.1) *может быть получено из жадного решения путем последовательности применений операций Вставка и Удаление.*

Доказательство. Пусть x_G — жадное, а x^* — оптимальное решения задачи (2.1). Пусть $a_\alpha \leq \cdots \leq a_1 < s$ — набор таких индексов, что $x_i^* = 0$, а $s \leq b_1 \leq \cdots \leq b_\beta$ — набор таких индексов, что $x_i^* = 1$.

Введем два индекса i, j. На начальном этапе положим $i = 1, j = 1$. Предлагаемый алгоритм последовательно применяет операции Вставка и Удаление, модифицируя жадное решение. Пусть на некотором шаге получено сбалансированное решение x. Если $w(x) > C$ и $i < \alpha$, то положим $x_{a_i} = 1$ и увеличим счетчик $i = i + 1$. Если $w(x) \leq C$ и $j < \beta$, то положим $x_{b_j} = 1$ и увеличим счетчик $j = j + 1$.

Рассмотрим различные варианты остановки алгоритма.

1. На очередном шаге получена ситуация $w(x) > C$ и $i = \alpha$. Если при этом $j = \beta$, то $x = x^*$. Если $j < \beta$, то $w(x^*) = w(x) + \sum_{k=j}^{\beta} w_k > w(x) > C$. Это противоречит тому, что x^* удовлетворяет ограничению задачи.

2. На очередном шаге получена ситуация $w(x) \leq C$ и $j = \beta$. Если при этом $i = \alpha$, то $x = x^*$. Если $i < \alpha$, то $p(x) = p(x^*) + \sum_{k=i}^{\alpha} p_k > p(x^*)$. Это противоречит тому, что x^* — оптимальное решение. $\square$.

Очевидна справедливость следующего вспомогательного утверждения.

Утверждение 9 Пусть $x' \in BF$ и $x'' \in BF$, $Core(x') \subseteq Core(x'')$, $w(x') = w(x'')$. Пусть $x'' \to x$, тогде $x' \to x$.

Рассмотрим алгоритм BALSUB [1], предназначенный для решения задачи о сумме подмножеств методом динамического программирования. Для хранения будем использовать таблицу, в ячейках которой находятся величины $g(b, w)$, определяемые соотношением:

$$g(b, \overline{w}) = \max\{a \in \overline{1, s} : \exists x, x \text{ is a balanced filling }, w(x) = \overline{w}\},$$

где $w(x) = \sum_{i=1}^{a-1} w_i + \sum_{i=a}^{b} w_i x_i$.

Пусть $\hat{w} = \sum_{i=1}^{s-1} w_i$ — вес жадного решения. Очевидно, что $g(b, \hat{w}) = s$.

цикл $w := c - w_{\max} + 1$ **до** c **выполнять**
| $g(s-1, w) := 0;$
конец
цикл $w := c + 1$ **до** $c + w_{\max}$ **выполнять**
| $g(s-1, w) := 1;$
конец
$g(s-1, \hat{w}) := s$
цикл $b := s$ **до** n **выполнять**
| **цикл** $w := c - w_{\max} + 1$ **до** $c + w_{\max}$ **выполнять**
| | $g(b, w) := g(b-1, w);$
| **конец**
| **цикл** $w := c - w_{\max} + 1$ **до** c **выполнять**
| | $w' = w + w_b$
| | $g(b, w') := \max(g(b, w'), g(b-1, w);$
| **конец**
| **цикл** $w := c + w_b$ **до** $c + 1$ **выполнять**
| | **цикл** $j := g(b-1, w)$ **до** $g(b, w) - 1$ **выполнять**
| | | $w' = w - w_j$
| | | $g(b, w') := \max(g(b, w'), j);$
| | **конец**
| **конец**
конец

Алгоритм 2: BALSUB algorithm

Утверждение 10 Число шагов в алгоритме BALSUB является величиной $O(nw_{\max})$.

Доказательство. Так как

$$\sum_{b=s+1}^{n} (g(b, w) - g(b-1, w)) = g(n, w) - g(s, w) \leq n,$$

то общее число шагов, очевидно составляет $O(nw_{\max})$. □

Пример 4 Решить задачу о сумме подмножеств:

$$6x_1 + 4x_2 + 2x_3 + 6x_4 + 4x_5 + 3x_3 \to \max$$
$$6x_1 + 4x_2 + 2x_3 + 6x_4 + 4x_5 + 3x_3 \leq 15.$$

w \ b	3	4	5	6
10	0	1	1	1
11	0	0	0	1
12	4	4	4	4
13	0	0	0	2
14	0	2	3	3
15	0	0	0	**4**
16	1	3	4	4
17	1	1	1	3
18	1	4	4	4
19	1	1	1	1
20	1	1	1	1
21	1	1	1	1

Несложно видеть, что полученное решение имеет вид 111001.

2.3 Задача об одномерном булевом ранце

2.3.1 Жадные алгоритмы

Опишем простейший жадный алгоритм. На первом шаге переменные сортируются в порядке невозрастающей эффективности:

$$\frac{p_1}{w_1} \geq \cdots \geq \frac{p_n}{w_n}.$$

Далее предметы укладываются в ранец в порядке увеличения индексов до тех пор, пока не нарушено ограничение по весу.

$c := C$
$p := 0$
цикл $i := 1$ **до** n **выполнять**
  Если $w_i < c$ **, тогда**
    $x_i := 1$
    $c := c - w_i$
    $p := p + p_i$
  иначе
    $x_i := 0$
  Конец условия
конец

Алгоритм 3: Algorithm KnapGreedy1

Обозначим полученное решение через x_{G1}, $z_{G1} = f(x_{G1})$. Заметим, что отношение $\frac{z_{G1}}{z_*}$ может быть сколь угодно малым. Достаточно рассмотреть

пример $p_1 = w_1 = 1, p_2 = w_2 = k$, $c = k$. В этом примере $z_G = 1, z_* = k$. Очевидно, что за счет увеличения k можно сделать отношение $\frac{z_{G1}}{z_*}$ сколь угодно малым.

Приведенный пример подсказывает на возможный путь повышения точности жадного алгоритма: следует выбирать максимальное значение из z_G и максимальной стоимости одного предмета. Псевдокод представлен на листинге 4

```
c := C
p := 0
i* := 1
p_m := p_1
цикл i := 1 до n выполнять
    Если w_i < c , тогда
        x_i := 1
        c := c − w_i
        p := p + p_i
    иначе
        x_i := 0
        Если w_i < C и p_i > p_m , тогда
            i* := i
            p_m := p_i
        Конец условия
    Конец условия
конец
Если p_m > p , тогда
    p := p_m
    цикл i := 1 до n выполнять
        x_i := 0
    конец
    x_{i*} := 1
Конец условия
```

Алгоритм 4: Algorithm KnapGreedy2

Заметим, что $z_{G2} = \max(z_{G1}, p_m)$. Очевидно $z_{G1} + p_m > z_*$. Следовательно, $z_* < 2z_{G2}$. Откуда получим $\frac{z_{G2}}{z_*} > \frac{1}{2}$. ТОчность оценки следует из рассмотрения примера $n = 3, p_1 = w_1 = 1, p_2 = w_2 = p_3 = w_3 = k, C = 2k$. Для этого примера $z_{G2} = 1 + k$, $z_* = 2k$. Отношение $\frac{z_{G2}}{z_*} = \frac{k+1}{k}$ стремится к $1/2$ с ростом k.

2.4 Приложения

2.4.1 Криптосистема Меркля-Хеллмана

В 1978 году Мерклем и Хеллманом было предложено использовать задачу о сумме подмножеств в качестве основы для шифрования сообщений. В качестве закрытого ключа используется супервозрастающая последовательность натуральных чисел $\{w_1, \ldots, w_n\}$, где

$$w_i > \sum_{j=1}^{i-1} w_j, i = 2, \ldots, n.$$

Заметим, что с таким набором коэффициентов задача о сумме подмножеств легко решается (алгоритм SUPERINCR).

$c := C$
цикл $i := n$ **до** 1 **выполнять**
  Если $w_i < c$ **, тогда**
    $x_i := 1$
    $c := c - w_i$
  иначе
    $x_i := 0$
  Конец условия
  Если $c = 0$ **, тогда**
    Возвратить *True*
  Конец условия
конец
Возвратить *False*

Алгоритм 5: Algorithm SUPERINCR

Супервозрастающая последовательность играет роль закрытого ключа, используемого для расшифровки сообщения. Для зашифровки используется открытый ключ, который получается из закрытого ключа следующим образом:

$$b_i := rw_i \mod q,$$

где $q > \sum_{i=1}^{n} w_i$, q и r — взаимно просты.

Утверждение 11 Задача о сумме подмножеств с супервозрастающей последовательностью коэффициентов $w_1, \ldots, w_n$ и правой частью C имеет решение тогда и только тогда, когда имеет решение задача с коэффициентами $b_i := rw_i \mod q, i = 1, \ldots, n$, $D = rC \mod q$, где $q > \sum_{i=1}^{n} w_i$, q и r — взаимно просты.

При этом коэффициенты b_i, вообще говоря, не являются супервозрастающей последовательностью, поэтому решение задачи с такими коэффициентами уже является тривиальной задачей.

Поскольку q и r — взаимно-просты, то существует единственный элемент r^{-1} кольца вычетов по модулю q такой что $rr^{-1} = 1 \mod q$. Процесс шифрования выглядит следующим образом. Пусть имеется двоичная последовательность $x_1, \ldots, x_n$, которую нужно зашифровать. Используя открытый ключ, вычисляется сумма $\hat{C} = \sum_{i=1}^{n} x_i b_i$. Кодом сообщения будет число $\hat{C}$.

На стороне приема вычисляется число $C = r^{-1}\hat{C} \mod q$ и решается (за линейное время) эквивалентная задача с закрытым ключом $w_1, \ldots, w_n$ и правой частью C.

Рассмотрим следующий пример. В качестве закрытого ключа рассмотрим рюкзак $S = (2, 3, 7, 14, 30, 57, 20, 251)$. Положим $r = 41$ и $q = 491$. Вычислим открытый ключ:

$$
\begin{aligned}
b_1 &= 2\,r = 2 \cdot 41 = 82 \mod 491 \\
b_2 &= 3\,r = 3 \cdot 41 = 123 \mod 491 \\
b_3 &= 7\,r = 7 \cdot 41 = 287 \mod 491 \\
b_4 &= 14\,r = 14 \cdot 41 = 83 \mod 491 \\
b_5 &= 30\,r = 30 \cdot 41 = 248 \mod 491 \\
b_6 &= 57\,r = 57 \cdot 41 = 373 \mod 491 \\
b_7 &= 120\,r = 120 \cdot 41 = 10 \mod 491 \\
b_8 &= 251\,r = 251 \cdot 41 = 471 \mod 491
\end{aligned}
$$

Несложно установить, что $r^{-1} = 12 \mod 491$

Предположим, что требуется зашифровать сообщение 10010110. Используя открытый ключ, получим:

$$\hat{C} = 2 + 83 + 373 + 10 = 548.$$

Для расшифровки будет использован закрытый ключ и $C = r^{-1}\hat{C} \mod q = 12 * 548 = 193 \mod 491$.

Корректность приведенной схемы следует из того, что если r и q взаимно просты, то из $ar = br \mod q$ следует, что $a = b \mod q$.

Глава 3

Задача об упаковке ящиков

3.1 Постановка задачи

Задача об упаковке ящиков, известная в англоязычной литературе под названием *bin-packing problem*, состоит в минимизации числа одинаковых ящиков, требуемых для упаковки заданного набора предметов. Математически эта задача может быть сформулирована как задача булева программирования. Пусть каждый из ящиков имеет вместимость C, и имеется n предметов с весами $w_1, \ldots, w_n$. Тривиальная верхняя граница на число используемых ящиков — число предметов n. Это значение целевой функции соответствует ситуации, когда каждый предмет кладется в отдельный ящик. Определим n булевых переменных y_i, $i = \overline{1,n}$. Переменная y_i принимает значение 1, если i-й ящик задействован (в нем есть хотя бы один предмет). Тогда целевая функция будет иметь вид

$$z = \sum_{i=1}^{n} y_i.$$

Введем переменные x_{ij}, принимающие значение 1, если i-й предмет размещен в ящике с номером j и 0 в противном случае. Задача имеет решение только при условии $w_i \leq C$ для всех $i \in \overline{1,n}$. Поскольку каждый предмет должен быть положен хотя бы в один ящик, то должны выполняться соотношения

$$\sum_{j=1}^{n} x_{ij} = 1, \ i \in \overline{1,n}. \tag{3.1}$$

Из необходимости соблюдения ограничения на суммарный вес предметов в ящике, получим неравенства:

$$\sum_{i=1}^{n} x_{ij} w_i \le y_j C,\ j \in \overline{1,n}. \tag{3.2}$$

Обобщая сказанное выше, можно выписать следующую формальную постановку задачи:

$$\begin{aligned}
&z = \sum_{i=1}^{n} y_i \to \max,\\
&\sum_{j=1}^{n} x_{ij} = 1,\ i \in \overline{1,n},\\
&\sum_{i=1}^{n} x_{ij} w_i \le y_j C,\ j \in \overline{1,n},\\
&y_i \in \{0,1\}, i \in \overline{1,n}\\
&x_{ij} \in \{0,1\}, i \in \overline{1,n}, j \in \overline{1,n}.
\end{aligned} \tag{3.3}$$

Для булева набора x_{ij}, $i \in \overline{1,n}$, $j \in \overline{1,m}$ обозначим через $v_j(x) = \sum_{i=1}^{n} x_{ij}$ число предметов в j-м ящике. Определим $b(x)$ число использованных ящиков. Очевидно $b(x) = |J|$, где $J = \{j \in \overline{1,m} : v_j(x) > 0\}$. Пусть $\underline{w} = \min_{i=1}^{n} w_i$. Число предметов в одном ящике не может превосходить величины $p = \lfloor C/\underline{w} \rfloor$. Введем коэффициенты $p_1, \ldots, p_m$, определяемые соотношениями

$$p_1 = 1, p_{j+1} = p\left(1 + \sum_{i=1}^{j} p_j\right), j = 2, \ldots, m-1. \tag{3.4}$$

Определим целевую фунцию $f(x) = \sum_{j=1}^{m} (p_j v_j)$.

Утверждение 12 Пусть x и $\hat{x}$ — два набора булевых переменных x_{ij}, $i \in \overline{1,n}$, $j \in \overline{1,m}$, удовлетворяющих ограничениям (3.1), (3.2). *Если $b(x) < b(\hat{x})$, то $f(x) < f(\hat{x})$.*

Доказательство. Пусть $b(x) < b(\hat{x})$. Тогда найдется такое $j_* \in \overline{1,m}$, что $v_{j_*}(\hat{x}) \ge 1$ и $v_j(x) = 0$ при $j \ge j_*$. Тогда

$$f(\hat{x}) \ge p_{j_*} v_{j_*} \ge p_{j_*} = p\left(1 + \sum_{i=1}^{j} p_j\right) > \sum_{i=1}^{j} p\, p_j \ge f(x).$$

Сформулируем теперь математическую постановку задачи об упаковке ящиков полностью:

$$\begin{aligned} & f(x) \to \min \text{ при условиях} \\ & \sum_{j=1}^{m} x_{ij} = 1,\ i \in \overline{1,n}, \\ & \sum_{i=1}^{n} x_{ij} w_i \le C,\ j \in \overline{1,m}, \\ & x_{ij} \in \{0,1\}. \end{aligned} \tag{3.5}$$

Эта задача имеет множество приложений. Известным приложением является задача разрезания листа на заготовки. Пусть имеется прямоугольный лист бумаги, который нужно разрезать на заготовки одинаковой ширины L, но различной высоты, отрезав от листа минимальный по длине прямоугольный фрагмент. Тогда высота листа будет играть роль ограничения на вместимость ящика C, а высоты заготовок — роль весов предметов. нужно минимизировать число отрезаемых полос ширины L. Каждая полоса заполняется заготовками по высоте.

3.2 Приближенные методы решения

3.2.1 Жадные алгоритмы

Простейший алгоритм **Next-Fit** для приближенного нахождения решения состоит в том, чтобы заполнять текущий ящик до тех пор, пока в нем есть место. Как только место заканчивается переходить к зполнению следующего ящика.

Утверждение 13 Пусть $NF(I)$ — значение целевой функции, найденное для задачи I алгоритмом $Next-Fit$, а $z(I)$ — оптимальное решение. Тогда

$$\frac{NF(I)}{z(I)} \le 2.$$

Доказательство. Пусть $NF(I) = k$. Пусть в результате работы алгоритма NF суммарный вес предметов в i-м ящике составляет c_i.

Очевидно, что:

$$\begin{aligned} c_1 + w_{i_1} &> C, \\ c_2 + w_{i_2} &> C, \\ &\dots \\ c_{k-1} + w_{i_{k-1}} &> C, \end{aligned}$$

где w_{i_j} — вес предмета, который не поместился в ящик j в процессе работы алгоритма NF. Так как $\sum_{i=1}^{k-1} c_i < W$ и $\sum_{j=1}^{k-1} w_{i_j} \leq W$, то суммируя неравенства получим:

$$(k-1) < \sum_{i=1}^{k-1} c_i + \sum_{j=1}^{k-1} w_{i_j} < 2W,$$

$$k - 1 < \frac{2W}{C},$$

$$k < \frac{2W}{C} + 1$$

Так как очевидно $z \geq \frac{W}{C}$, то

$$k < 2z + 1.$$

так как k — целое, то $k \leq 2z$.

Суммируя эти неравенства и добавляя $w_1 + w_k$, получим

$$2W > (k-1)C.$$

Отсюда $k < \frac{2W}{C} + 1 \leq \lceil \frac{2W}{C} \rceil + 1$. Так как k — целое число, то $k \leq \lceil \frac{2W}{C} \rceil \leq 2\lceil \frac{W}{C} \rceil$. Очевидно, что $z_I \geq \lceil \frac{W}{C} \rceil$. Это доказывает утверждение.

Упражнение 8 Показать, что для любого ε найдется пример I_ε, такой что

$$\frac{NF(I)}{z(I_\varepsilon)} > 2 - \varepsilon.$$

Другой алгоритм **First-Fit** состоит в том, чтобы добавлять очередной предмет в корзину с наименьшим номером, в которую он помещается. Если такой не находится из числа заполненных корзин, то берется новая корзина. Можно показать, что для данного алгоритма имеет место соотношение:

$$\frac{17}{10} z(I_\varepsilon) - 8 \leq FF(I) \leq \frac{17}{10} z(I_\varepsilon) + 2.$$

Введем понятие *асимптотической точности* для некоторого приближенного алгоритма A как такого минимального действительного числа $r^{\infty}(A)$, что для любого $\varepsilon > 0$ найдется некоторое $k_{\varepsilon} \in \mathbb{Z}^{+}$, такое что

$$\frac{A(I)}{z(I)} \leq r^{\infty}(A) + \varepsilon \text{ для любой задачи } I, z(I) > k_{\varepsilon}.$$

Очевидно, $r^{\infty}(NF) = 2$, $r^{\infty}(FF) = \frac{17}{10}$.

Рассмотренные алгоритмы можно дополнить предварительной сортировкой предметов по убыванию веса. При этом точность увеличится:

$r^{\infty}(SNF) \simeq 1,691$, $r^{\infty}(SFF) \simeq 1,222$.

3.3 Нижние оценки

Рассмотренные в предыдущем пункте приближенные алгоритмы позволяют получить верхнюю оценку целевой функции в задаче об упаковке контейнеров. В этом разделе рассмотрим вычисление нижних оценок. Нижней оценкой для подзадачи I назовем число $L(I) \leq z(I)$. *Точностью наихудшего случая* для нижней оценки L называется величина $\rho(L)$, определяемая следующим образом:

$$\rho(L) = \inf \frac{L(I)}{z(I)} \text{ по всем возможным задачам } I.$$

Простейшая оценка вычисляется по очевидной формуле $L_1(I) = \left\lceil \frac{\sum_{i=1}^{n} w_i}{C} \right\rceil$. Содержательно эта оценка означает, что ящиков не может быть меньше, чем суммарный вес предметов, деленный на вместимость одного ящика.

Утверждение 14

$$\rho(L_1) = \frac{1}{2}.$$

Доказательство. Рассмотрим произвольную задачу об упаковке контейнеров. Заметим, что в оптимальном решении не может быть двух корзин, заполненных не более, чем на половину. В противном случае эти корзины могут быть объединены в одну. Поэтому, в отпимальном решении, все корзины, кроме, может быть одной, заполнены более чем на половину. Поэтому

$$\sum_{i=1}^{n} w_i > (z(I) - 1)\frac{C}{2}.$$

Отсюда получим

$$z(I) < 2\left(\frac{\sum_{i=1}^{n} w_i}{C}\right) + 1.$$

Так как $z(I)$ — целое число, то

$$z(I) \leq \left\lceil 2\left(\frac{\sum_{i=1}^{n} w_i}{C}\right)\right\rceil \leq 2\left\lceil \frac{\sum_{i=1}^{n} w_i}{C}\right\rceil = 2L_1(I).$$

Таким образом $\frac{L_1(I)}{z(I)} \geq \frac{1}{2}$ для любой задачи I.

Теперь покажем, что существуют задачи для которых $\frac{L(I)}{z(I)}$ сколь угодно близко к $\frac{1}{2}$. Рассмотрим задачу I, где $w_j = k+1$ для всех $j \in \overline{1,n}$, $C = 2k$. Очевидно, $z(I) = n$. При этом, $L_1(I) = \left\lceil \frac{n(k+1)}{2k}\right\rceil$. При k стремящемся к бесконечности выражение $\frac{L_1(I)}{z(I)} \to \frac{1}{2}$. □

Оценка L_1 хорошо работает в ситуации, когда веса предметов примерно одинаковые. Определим другую, более сильную оценку. Пусть a — некоторое натуральное число, $0 \leq a \leq \frac{C}{2}$. Определим множества:

$$J_1 = \{j : w_j > C - a\},$$
$$J_2 = \{j : C - a \geq w_j > \frac{C}{2}\},$$
$$J_3 = \{j : \frac{C}{2} \geq w_j \geq a\}.$$

Утверждение 15 Выражение

$$L(a) = |J_1| + |J_2| + \max\left(0, \left\lceil \frac{\sum_{j\in J_3} w_j - (|J_2|C - \sum_{j\in J_2} w_j)}{C}\right\rceil\right).$$

является нижней границей для $z(I)$.

Доказательство. Для упаковки предметов из множества $J_1 \cup J_2$ нужно как минимум $|J_1| + |J_2|$ ящиков, т.к. вес каждого из этих предментов не меньше $c/2$, значит никакие два предмета из этого набора не могут быть одновременно положены в один ящик. Никакой предмет из множества J_3 не может быть положен в ящик, в котором уже лежит предмет из J_1. Общий объем свободного пространства, в ящиках, содержащих предметы из J_2, составляет $|J_2|C - \sum_{j\in J_2} w_j$. Если все это пространство будет заполнено предметами из множества J_3, то вес оставшихся предметов из множества J_3 составит $rw = \sum_{j\in J_3} w_j - (|J_2|C - \sum_{j\in J_2} w_j)$. На упаковку этих предметов потребуется не менее $\lceil \frac{rw}{C}\rceil$ ящиков. Утверждение доказано. □

Утверждение 16

$$L_2 \geq L_1.$$

Доказательство. Положим $a = 0$. Тогда

$$\begin{aligned}
L(a) &= 0 + |J_2| + \max\left(0, \left\lceil \frac{\sum_{j\in J_3} w_j - (|J_2|C - \sum_{j\in J_2} w_j)}{C} \right\rceil\right) \\
&= |J_2| + \max\left(0, \left\lceil \frac{\sum_{j\in \overline{1,n}} w_j - |J_2|C)}{C} \right\rceil\right) \\
&= |J_2| + \max\left(0, \left\lceil \frac{\sum_{j\in \overline{1,n}} w_j}{C} - |J_2| \right\rceil\right) \\
&= |J_2| + \max\left(0, \left\lceil \frac{\sum_{j\in \overline{1,n}} w_j}{C} \right\rceil - |J_2|\right) \\
&= |J_2| + \max\left(0, L_1 - |J_2|\right) \\
&= \max\left(|J_2|, L_1\right).
\end{aligned}$$

Вычисление оценки L_2 может быть сделано полным перебором по числам $\overline{0, C/2}$. Докажем справедливость следующего утверждения, которое облегчает вычисление оценки L_2.

Утверждение 17 Пусть V — список различных значений коэффициентов $w_1, \ldots, w_n$, не превосходящих $C/2$. Тогда

$$L_2 = \begin{cases} n, V = \emptyset, \\ \max_{a\in V} L(a), V \neq \emptyset. \end{cases}$$

Доказательство. Если $V = \emptyset$, то поскольку никакие два предмета не могут быть положены в один ящик, то потребуется не менее n ящиков. Рассмотрим случай $V \neq \emptyset$. Пусть a_1 и a_2 — два числа из диапазона $\overline{0, C/2}$, такие что $a_1 < a_2$, при этом множества J_3 для них одинаковы. Заметим, что

$$r(a) = |J_2|C - \sum_{j\in J_2(a)} w_j = \sum_{j\in J_2(a)} (C - w_j)$$

является невозрастающей функцией от a, так как $J_2(a_2) \subseteq J_2(a_1)$ при $a_1 < a_2$. Так как $|J_1(a)| + |J_2(a)|$ не зависит от a, $J_3(a_1) = J_3(a_2)$ и $r(a_1) \geq r(a_2)$, то $L(a_1) \leq L(a_2)$. Пусть веса предметов упорядочены по убыванию $w_1 \geq \cdots \geq w_n$. Заметим, что веса $w_i, w_i - 1, \ldots, w_{i+1} + 1$ порождают одинаковые множества J_3. При этом $L(w_i) \geq L(w_i - 1) \geq \cdots \geq L(w_{i+1} + 1)$. Тем самым утверждение доказано. □.

Таким образом, оценку L_2 можно вычислять, перебирая все различные коэффицинеты, не превосходящие $C/2$.

3.4 Методы решения

Рассмотрим теперь другую возможную постановку данной задачи, более удобную с практической точки зрения. Введем целочисленные переменные x_i, $i \in \overline{1,n}$ определяющие распределение предметов по ящикам: $x_i = j$, если предмет с номером i помещен в ящик j. Целевая функция, которую при этом нужно минимизировать, определяется следующим образом:

$$f(x) = \max_{i \in \overline{1,n}} x_i.$$

Ограничения на вместимость ящиков принимают следующий вид:

$$\sum_{i=1}^{n} \theta(x_i, j) w_i \leq C, j \in \overline{1,m},$$

где $\theta(i,j) = \begin{cases} 0, \text{ если } i = j, \\ 1, \text{ если } i \neq j. \end{cases}$

Таким образом получаем следующую математическую формулировку:

$$\begin{aligned} & f(x) = \max_{i \in \overline{1,m}} x_i \to \min \text{ при условиях} \\ & \sum_{i=1}^{n} \theta(x_i, j) w_i \leq C, j \in \overline{1,m}, \\ & 1 \leq x_i \leq m, \\ & x_i \in \mathbb{Z}. \end{aligned} \tag{3.6}$$

Предлагаемый алгоритм основан на методе ветвей и границ. Подзадача задачи (3.6) определяется присвоением части переменных x_i некоторых значений α_i:

$$\begin{aligned} & f(x) = \max_{i \in \overline{1,n}} x_i \to \min \text{ при условиях} \\ & \sum_{i=1}^{n} \theta(x_i, j) w_i \leq C, j \in \overline{1,n}, \\ & 1 \leq x_i \leq m, \\ & x_i = \alpha_i, i \in A \subseteq \overline{1,n}, \\ & x_i \in \mathbb{Z}. \end{aligned} \tag{3.7}$$

Для подзадачи s будем обозначать через $s.\alpha$ вектор длины n, определяющий частичное размещение предметов по ящикам. Будем полагать,

что если k-й элемент вектора содержит нулевое значение, то на размещение k-го предмета не наложено ограничений. В противном случае данный элемент содержит номер ящика, в который предмет размещен, т.е. имеет место ограничение $x_k = \alpha_k$. Обозначим через $f(s)$ величину $\max\{x_i : i \in A\}$ — максимальный номер использованного ящика. При фиксации значений всех переменных, т.е. когда $|A| = n$, мы получаем решение, значение целевой функции на котором совпадает с $f(s)$.

Для дальнейшего изложения определим еще один атрибут подзадачи s — вектор $s.r$ длины m, j-й элемент которого содержит остаточную вместимость ящика с номером j после размещения в нем назначенных предметов: $s.r_j = C - \sum_{i \in A} \theta(\alpha_i, j) w_i$.

Определим два правила отсева, позволяющие исключить задачу из дальнейшего рассмотрения.

Правило С1. Несовместность ограничений, т.е. если для некоторого $j \in \overline{1,m}$ справедливо $\sum_{i \in A} \theta(\alpha_i, j) w_i > C$. Данное условие можно записать как $s.r_j < 0$.

Правило С2. Решение данной подзадачи не может привести к улучшению рекорда, т.е. наилучшего найденного на данном шаге решения: $L(s) \geq f_r$. В качестве $L(s)$ рассматривается одна из приведенных в предыдущем пункте нижних границ.

Декомпозиция подзадачи осуществляется следующим образом. Берется следующий из предметов, которые еще не размещены по ящикам с номером $j = \max\{i : i \in A\} + 1$. Формируется множество подзадач, соответствующих возможным размещениям предмета с номером j по ящикам. Заметим, что при этом целесообразно сразу же проверить правила **С1** и **С2**. В случае выполнения хотя бы одного из них подзадача сразу же исключается из рассмотрения и не подвергается дальнейшей обработке.

При декомпозиции из подзадачи P получается b новых подзадач, получаемых присвоением переменной x_j одного из значений в диапазоне $\overline{1,b}$.

Как несложно заметить, любое оптимальное размещение предметов по ящикам не может содержать пустых ящиков в середине. Другими словами, для оптимального решения x в задаче (3.6) для любого $j : 1 \leq j \leq \max_{i \in \overline{1,n}} x_i$ найдется номер $i \in \overline{1,n}$, такой что $x_i = j$. Поскольку все ящики равноценны, то любую оптимальную упаковку можно получить последовательным размещением предметов таким образом, чтобы ящик с номером j выбирался только, если среди ящиков с меньшим номером нет пустых. Поэтому можно без ограничения общности положить

$$b = \min(f_r - 1, f(s) + 1).$$

Псевдокод алгоритма Binpack, выполняющий последовательный ва-

риант метода ветвей и границ для задачи о сумме подмножеств, приведен ниже.

Исходные параметры:
S — множество подзадач
x_r — рекордное решение
f_r — значение рекорда
Результат:
x_r — рекордное решение
f_r — значение рекорда
Пока $S \neq \emptyset$ **:**
  select s from S
  $S := S \setminus \{s\}$
  Если $L(s) < f_r$ **, тогда**
    $b := \min(f_r - 1, f(s) + 1)$
    $k := 1 + \max\{i : s.\alpha_i \neq 0\}$
    цикл $j = 1$ **до** b **выполнять**
      Если $s.r_j \geq w_k$ **, тогда**
        create new subproblem s'
        $s'.\alpha := s.\alpha$
        $s'.\alpha_k := j$
        $s'.r := s.r$
        $s'.r_j := s.r_j - w_k$
        Если $k = n$ **, тогда**
          Если $f(s') < f_r$ **, тогда**
            $x_r := s'.\alpha$
            $f_r := f(s')$
          Конец условия
        иначе
          $S := S \cup \{s'\}$
        Конец условия
      Конец условия
    конец
  Конец условия
Конец цикла
return (x_r, f_r)

Алгоритм 6: Binpack

Входными данными для него являются множество подзадач, рекордное решение и значение рекорда. В базовом варианте перед началом работы исходное множество подзадач состоит из одного элемента — начальной задачи. Рекорд может быть найден с помощью любого эвристического алгоритма. Алгоритм завершается когда множество подзадач

становится пустым.

Глава 4

Задачи теории расписаний

4.1 Расписания на одной машине

Рассмотрим ситуацию, когда имеется ровно одна машина для выполнения набора n работ, каждая характеризуется временем выполнения p_i и важностью w_i. Необходимо определить такую последовательность работ, при которой минимизируется взвешенное время завершения:

$$\sum_{i=1}^{n} w_i C_i \to \min, \tag{4.1}$$

где C_i — время завершения работы i.

Утверждение 18 Оптимальным расписанием для задачи 4.1 будет расписание, при котором работы упорядочены по не-возрастанию величин w_i/p_i.

Доказательство. Предположим противное. Тогда в оптимальном расписании найдутся две работы с номерами j и k, расположенные подряд в расписании, что $\frac{w_j}{p_j} < \frac{w_k}{p_k}$. Предположим, что обработка работы j начата в момент времени t. Тогда работа j завершилась в момент $t+p_j$, а работа k в момент $t+p_j+p_k$. Поменяем местами работы j и k. Значение целевой функции z' для полученного расписания связано со значением целевой функцией исходного расписания z соотношением:

$$\begin{aligned} z' &= z - ((t+p_j)w_j + (t+p_j+p_k)w_k) + ((t+p_k)w_k + (t+p_k+p_j)w_j), \\ &= z - p_j w_k + p_k w_j. \end{aligned}$$

Так как $\frac{w_j}{p_j} < \frac{w_k}{p_k}$, то $p_k w_j < p_j w_k$. Следовательно, $-p_j w_k + w_j p_k < 0$ и $z' < z$. Это противоречит оптимальности исходного расписания.

Работы могут быть связаны отношением предшествования. Рассмотрим *цепное отношение предшествования*, когда множество работ является объединением некоторого конечного числа цепей. Работы в цепи должны выполняться в порядке следования.

Начнем с ситуации, когда выполнение цепи работ не должно прерываться, т.е. между работами одной цепи не должны выполняться операции другой цепи. Тогда справедливо следующее утверждение.

Утверждение 19 Если

$$\frac{\sum_{i=1}^{k} w_i}{\sum_{i=1}^{k} p_i} > \frac{\sum_{i=k+1}^{n} w_i}{\sum_{i=k+1}^{n} p_i}, \tag{4.2}$$

то в оптимальном расписании цепь $1, \ldots, k$ должна следовать до цепи $k+1, \ldots, n$.

Доказательство. Рассмотрим оба варианта.

$$z = \sum_{i=1}^{k} \left(w_i \sum_{j=1}^{i} p_j \right) + \sum_{i=k+1}^{n} \left(w_i \left(\sum_{j=1}^{k} p_j + \sum_{j=k+1}^{i} p_j \right) \right),$$

$$z' = \sum_{i=k+1}^{n} \left(w_i \sum_{j=k+1}^{i} p_j \right) + \sum_{i=1}^{k} \left(w_i \left(\sum_{j=k+1}^{n} p_j + \sum_{j=1}^{i} p_j \right) \right).$$

Заметим, что

$$\Delta = z - z' = \left(\sum_{j=1}^{k} p_j \right) \left(\sum_{i=k+1}^{n} w_i \right) - \left(\sum_{j=k+1}^{n} p_j \right) \left(\sum_{i=1}^{k} w_i \right).$$

В силу 4.2 $\Delta < 0$. Поэтому $z < z'$. $\square$

Глава 5

Целочисленное линейное программирование

5.1 Основы линейного программирования

5.1.1 Общие сведения о задачах линейного программирования

Стандартная форма записи задачи линейного программирования:

$$\begin{aligned} c^T x &\to \min, \\ Ax = b,\ A : m \times n, & \\ x &\geq 0. \end{aligned} \tag{5.1}$$

Рассмотрим задачу в стандартной форме (5.1). *Базисом* матрицы A называется набор, состоящий из m линейно независимых столбцов. *Базисным решением* назовем вектор, у которого в компонентах с номерами столбцов из базиса находятся элементы вектора $B^{-1}b$, а остальные компоненты равны 0. Если предположить, что базис соответствует первым m столбцам матрицы A, базисное решение будет иметь вид

$x = \begin{pmatrix} B^{-1}b \\ 0 \end{pmatrix}$

В дальнейшем сделаем ряд предположений.

Предположение 1. Матрица A имеет ранг m.

Предположение 2. Множество допустимых точек не пусто.

Утверждение 20 При предположениях 1,2 существует по меньшей мере одно базисное допустимое решение.

Предположение 3. Множество $\{c^T x : x \in F\}$ ограничено снизу.
Каноническая форма записи задачи линейного программирования:

$$\begin{pmatrix} 1 & 0 & c^T \\ 0 & I & \hat{A} \end{pmatrix} \times \begin{pmatrix} -z \\ x_B \\ x_N \end{pmatrix} = \begin{pmatrix} -z_0 \\ \hat{b} \end{pmatrix} \tag{5.2}$$

Решение, получаемое подстановкой 0 вместо компонентов x_N, и элементов вектора $\hat{b}$ вместо x_B будет, очевидно, базисным для задачи (5.5).

Базисное решение называется *допустимым*, если оно удовлетворяет ограничениям $x_i \geq 0$, $i = 1, \ldots, n$.

Линейное подпространство R^n — множество решений однородной системы алгебраических уравнений вида $Ax = b$. Размерностью множества $S \subseteq R^n$ называется наименьшая размерность афинного подпространства R^n , содержащего S.

Гиперплоскостью называется аффинное подпространство размерностью $n - 1$. Гиперплоскость задается уравлением $a^T x + b = 0$, где не все компоненты вектора a равны нулю. Полупространством называется множество решений неравенства $a^T x + b \leq 0$.

Многогранником называется пересечение конечного числа полупространств, если это множество ограничено.

Рассмотрим гиперплоскость H, задаваемую уравнением $a^T x + b = 0$ и соответствующее полупространство HS, задаваемое уравнением $a^T x + b \leq 0$. Пусть P — некоторый многогранник. Пусть $Q \cap HS$. Если $Q \neq \emptyset$ и $Q = P \cap H$, то Q называется *гранью многогранника* P.

Вершина — грань размерности 0, *ребро* — грань размерности 1, *гиперграньь* — грань размерности $n - 1$.

Теорема 4 Любой многогранник являетяс выпуклой комбинацией своих вершин. Выпуклая комбинация конечного числа точек является многогранником.

Любой многогранник может быть задан условиями:

$$Ax = b, x \geq 0.$$

Теорема 5 Пусть P — многогранник, определяемый соотношением $Ax = b, x \geq 0$. Тогда следующие утверждения эквивалентны:

1. *x является вершиной P;*
2. *x не может быть строго выпуклой комбинацией двух других точек P;*

3. x — базовое допустимое решение системы $Ax = b, x \geq 0$.

Теорема 6 В любой задаче линейного программирования, удовлетворяющей предположениям 1-3, имеется оптимальное базовое допустимое решение.

Пусть дано БДР $x = \begin{pmatrix} x_B \\ x_N \end{pmatrix}$, соответствующее базису $B = A(1), \ldots, A(m)$. Тогда справедливо равенство:

$$Bx_B = b. \tag{5.3}$$

Пусть $j > m$ и j-й столбец $A(j)$ матрицы A может быть представлен в виде линейной комбинации базисных столбцов $A(1), \ldots, A(m)$ с коэффициентами d_i т.е.:

$$Bd = A(j). \tag{5.4}$$

Умножая равенство (5.4) на некоторое $\alpha > 0$ получим

$$B(x_B - \alpha d) + \alpha A(j) = b.$$

Решение x', где $x'_i = \begin{cases} x_i - \alpha d_i, i \neq j, 1 \leq i \leq m \\ \alpha, i = j, \\ 0, i > m, i \neq j. \end{cases}$ будет очевидно удовлетворять ограничению задачи ЛП. При этом оно будет допустимым, если $x_i - \alpha d_i \geq 0$ для всех $i \in \overline{1, m}$. Т.е. $\alpha < \min_{d_i > 0} x_i/d_i$.

Замечение 1. Если все $d_i < 0$, то многогранник, соответствующей задаче ЛП неограничен.

Переход от одного БДР к другому осуществляется согласно следующей теореме.

Теорема 7 Пусть дано БДР x, соответствующее базису $B = A(1), \ldots, A(m)$. Пусть $j > m$ и j-й столбец $A(j)$ матрицы A может быть представлен в виде линейной комбинации базисных столбцов $A(1), \ldots, A(m)$ с коэффициентами d_i: $A(j) = Bd$. Тогда решение определяется равенствами:

$$\alpha = \min_{d_i > 0}(x_i/d_i) = x_l/d_l$$

$$x'_i = \begin{cases} x_i - \alpha d_i, i \neq l, \\ \alpha, i = l. \end{cases}$$

будет базисным с базисом B', определяемым равенствами:

$$B'(i) = \begin{cases} B(i), i \neq l, \\ j, i = l. \end{cases}$$

Замечение 2. Если минимум $\min_{d_i>0}(x_i/d_i) = x_l/d_l$ достигается при нескольких i, то получаемое в результате решение задачи ЛП вырождено, т.к. содержит более $n - m$ нулевых компонентов.

5.1.2 Симплекс-метод

Рассмотрим задачу ЛП в канонической форме

$$\begin{pmatrix} 1 & 0 & c^T \\ 0 & I & A \end{pmatrix} \times \begin{pmatrix} -z \\ x_B \\ x_N \end{pmatrix} = \begin{pmatrix} -z_0 \\ b \end{pmatrix} \tag{5.5}$$

Здесь z — значение целевой функции, удовлетворяющее уравнению

$$-z + c^T x_N = -z_0. \tag{5.6}$$

Так как значения небазисных переменных равны 0, то оптимальное значение целевой функции есть z_0.

Заметим, что если $c \geq 0$, то найдено оптимальное решение задачи $x_B = b, x_N = 0$. Действительно, любое другое решение имеет неотрицательные компоненты x_N из которых хотя бы одна положительна, поэтому из (5.6) следует, что значение $c^T x$ будет положительно, а $z = z_0 + c^T x > z_0$.

Пусть столбец $A(j)$ вводится в базис, а столбец $B(l)$ выводится из базиса. Индекс выводимого из базиса столбца всегда выбирается так, чтобы выполнялось $c_l < 0$. Тогда для обнуления соответствующего элемента c_l к $-z_0$ будет добавлена величина $-c_l b_j/v$, где v — значение ведущего элемента. Так как $c_l < 0$, то это означает, что $-z_0$ увеличиться, а значение целевой функции z_0 соответственно уменьшится.

Пример 5 Решить задачу ЛП, заданную в стандартной форме:

$$\begin{aligned} z = x_1 + x_2 + x_3 + x_4 + x_5 \to \max, \\ 3x_1 + 2x_2 + x_3 = 1, \\ 5x_1 + x_2 + x_3 + x_4 = 3, \\ 2x_1 + 5x_2 + x_3 + x_5 = 4. \end{aligned}$$

Составим симплекс-таблицу:

1	1	1	1	1	0
3	2	1	0	0	1
5	1	1	1	0	3
2	5	1	0	1	4

Несложно заметить, что столбцы $3, 4, 5$ образуют базис. Приведем таблицу к каноническому виду:

-3	-3	0	0	0	-6
3	**2**	1	0	0	1
2	-1	0	1	0	2
-1	3	0	0	1	3

Таким образом получили базовое решение со значением целевой функции 6. Для введения в базис можно выбрать столбец с индексом 1 либо 2, т.к. соответствующий коэффициент c_j отрицателен. Возьмем столбец с номером 2. В качестве ведущего возьмем первый элемент, т.к. для него минимально отношение b_i/a_{ij}.

$3/2$	0	$3/2$	0	0	$-9/2$
$3/2$	1	$1/2$	0	0	$1/2$
$7/2$	0	$1/2$	1	0	$5/2$
$-11/2$	0	$-3/2$	0	1	$3/2$

Тем самым получено новое допустимое базовое решение, которое при этом является оптимальным, поскольку все коэффициенты c_j неотрицательны. Значение целевой функции при этом 4.5.

5.1.3 Двойственный симплекс-метод

В двойственном симплекс-методе матрица приводится к канонической форме, когда правая часть равенств может быть отрицательной, но все коэффициенты c_j должны быть неотрицательны. В конечном итоге требуется получить неотрицательный правый столбец, сохраняя положительность коэффициентов c_j.

Пока $\min_{i=1,\dots,m} b_i < 0$ **:**
  $r := \operatorname{argmin}_{i=1,\dots,m} b_i$
  Если $\min_{i=1,\dots,n} a_{ri} \geq 0$ **, тогда**
    задача несовместна
  Конец условия
  $s := \operatorname{argmin}_{a_{ri}<0} -(c_i/a_{ri})$
  ввести в базис переменную x_s
Конец цикла

Алгоритм 7: Двойственный симплекс-метод

Пример 6 Решить задачу в следующей канонической форме двойственным симплекс-методом

$$\begin{aligned} -z + x_4 + 2x_5 + 4x_6 &= -5 \\ x_1 + 2x_4 - 5x_5 + 5x_6 &= 9 \\ x_2 - 3x_4 + 6x_5 - 3x_6 &= -3 \\ x_3 + x_4 - 4x_5 + 2x_6 &= 2 \end{aligned}$$

Задача записана в канонической форме. Соответствуюшее базисное решение недопустимо, поскольку в правой части присутствуют отрицательные элементы. Поскольку все коэффициенты в целевой функции неотрицательны, то можно применить двойственный симплекс-метод. Запишем симплекс-таблицу:

0	0	0	1	2	4	-5
1	0	0	2	-5	5	9
0	1	0	$\mathbf{-3}$	6	-3	-3
0	0	1	1	-4	2	2

Выбираем строку с минимальным отрицательными элементом и столбец исходя из критерия $argmin\{-c_j/a_{ij},\, a_{ij} < 0\}$

0	1/3	0	0	4	3	-6
1	2/3	0	0	-1	3	7
0	$-1/3$	0	1	-2	1	1
0	1/3	1	0	-2	1	1

Теперь задача находится в канонической форме, и соответствующее базисное решение не является оптимальным, т.к. $b \geq 0$, $c \geq 0$.

5.2 Метод отсечений Гомори

Докажем вспомогательное утверждение.

Утверждение 21 Пусть справедливо равенство

$$\sum_{i=1}^{n}(I_i + F_i)x_i = I_{n+1} + F_{n+1}, \tag{5.7}$$

где $x_i \geq 0$, $x_i \in \mathbb{Z}$, $i = 1, \ldots, n$, $I_i \in \mathbb{Z}$, $i = 1, \ldots, n+1$, $0 \leq F_i < 1$, $i = 1, \ldots, n+1$. *Тогда*

$$\sum_{i=1}^{n} F_i x_i - F_{n+1} \in \mathbb{Z}, \tag{5.8}$$

$$\sum_{i=1}^{n} F_i x_i - F_{n+1} \geq 0. \tag{5.9}$$

Доказательство. Перепишем равенство (5.7) в следующем виде:

$$\sum_{i=1}^{n} I_i x_i - I_{n+1} = F_{n+1} - \sum_{i=1}^{n} F_i x_i.$$

Из этого соотношения непосредственно следует (5.8). Т.к. $F_{n+1} < 1$, а $\sum_{i=1}^{n} F_i x_i \geq 0$, то, поскольку $\sum_{i=1}^{n} I_i x_i - I_{n+1} \in \mathbb{Z}$, то $\sum_{i=1}^{n} I_i x_i - I_{n+1} \leq 0$. Следовательно $F_{n+1} - \sum_{i=1}^{n} F_i x_i \leq 0$. Отсюда непосредственно вытекает (5.9). □

Метод отсечений Гомори применяется к задаче целочисленного линейного программирования, заданной в стандартной форме (5.1). Требуется найти целочисленное решение. Задача сначала решается прямым симплекс-методом. Если полученное оптимальное решение целочисленно, то задача решена. В противном случае, выбирается уравнение с дробной правой частью: $a_{i1}x_1 + \cdots + a_{in}x_n = b_i$, $b_i \notin \mathbb{Z}$. Опираясь на утверждение 21, можно ввести следующее дополнительное ограничение в задачу:

$$\{a_{i1}\}x_1 + \cdots + \{a_{in}\}x_n - \{b_i\} \geq 0,$$

где через $\{a\}$ обозначена дробная часть числа a. Вводя дополнительную переменную $x_{n+1} \geq 0$, переходим к равенству

$$\{a_{i1}\}x_1 + \cdots + \{a_{in}\}x_n - x_{n+1} = \{b_i\}.$$

Из (5.9) следует, что $x_{n+1} \in \mathbb{Z}^+$.

Пример 7 Решить задачу ЦЛП

$$\begin{aligned} & x_2 \to \max, \\ & 3x_1 + 2x_2 \le 6, \\ & -3x_1 + 2x_2 \le 0, \\ & x_i \in \mathbb{Z}^+ \end{aligned} \tag{5.10}$$

Вводя дополнительные переменные, преобразуем задачу (5.2) к виду

$$\begin{aligned} & -z - x_2 = 0, \\ & 3x_1 + 2x_2 + x_3 = 6, \\ & -3x_1 + 2x_2 + x_4 = 0, \\ & x_i \in \mathbb{Z}^+ \end{aligned}$$

Соответствующая таблица имеет вид

x_1	x_2	x_3	x_4	
0	-1	0	0	0
3	**2**	1	0	6
-3	2	0	1	0

Задача записана в канонической форме. Применим прямой симплекс-метод.

x_1	x_2	x_3	x_4	
$-3/2$	0	0	$1/2$	0
6	0	1	-1	6
$-3/2$	1	0	$1/2$	0

x_1	x_2	x_3	x_4	
0	0	$1/4$	$1/4$	$3/2$
1	0	$1/6$	$-1/6$	1
0	1	$1/4$	$1/4$	$3/2$

Получена каноническая форма для допустимого базисного решения, которое очевидно не является целочисленным. Выполним отсечение Гомори на основе последнего уравнения.

x_1	x_2	x_3	x_4	x_5	
0	0	$1/4$	$1/4$	0	$3/2$
1	0	$1/6$	$-1/6$	0	1
0	1	$1/4$	$1/4$	0	$3/2$
0	0	$1/4$	$1/4$	-1	$1/2$

Домножая последнее уравнение на -1, получим каноническую форму:

x_1	x_2	x_3	x_4	x_5	
0	0	1/4	1/4	0	3/2
1	0	1/6	$-1/6$	0	1
0	1	1/4	1/4	0	3/2
0	0	**-1/4**	$-1/4$	1	$-1/2$

Данная каноническая форма соответствует недопустимому базисному решению. Применим двойственный симплекс-метод.

x_1	x_2	x_3	x_4	x_5	
0	0	0	0	1	1
1	0	0	$-1/3$	2/3	2/3
0	1	0	0	1	1
0	0	1	1	-4	2

Найдено оптимальное решение линейной задачи. Оно опять не является целочисленным. Строим отсечение Гомори по второму уравнению:

x_1	x_2	x_3	x_4	x_5	x_6	
0	0	0	0	1	0	1
1	0	0	$-1/3$	2/3	0	2/3
0	0	0	$-2/3$	$-2/3$	1	$-2/3$
0	1	0	0	1	0	1
0	0	1	1	-4	0	2

Применяем двойственный симплекс-метод.

x_1	x_2	x_3	x_4	x_5	x_6	
0	0	0	0	1	0	1
1	0	0	0	1	$-1/2$	1
0	0	0	1	1	$-3/2$	1
0	1	0	0	1	0	1
0	0	1	0	-5	3/2	1

Получена каноническая форма, соответствующая оптимальному решению задачи $x_1 = x_2 = 1$, т.к. правая часть содержит только целые числа.

Глава 6

Контрольные вопросы и упражнения

1. Классификация задач оптимизации по числу критериев, локальности экстремума, структуре допустимого множества. Классификация методов оптимизации: детерминированные и эвристические методы решения задач оптимизации.
2. Понятия алфавита, языка и машины Тьюринга. Примеры машин Тьюринга.
3. Недетерминированная машина Тьюринга, классы сложности P и NP. Проблема равенства классов. Примеры задач.
4. Полиномиальная сводимость, теорема Кука. Понятие NP-полной задачи.
5. NP-полнота задачи о 3-выполнимости. Преобразование произвольной КНФ к 3-КНФ.
6. NP-полнота задачи о трехмерном сочетании.
7. NP-полнота задачи о сумме подмножеств в форме распознавания.
8. NP-полнота в сильном смысле.
9. Задача о ранце, задача о сумме подмножеств: формулировка, основные понятия, экономическая интерпретация. Линейная релаксация.
10. Метод ветвей и границ в задаче о ранце, понятия подзадачи, верхней и нижней оценок. Общая схема алгоритма. Алгоритмическая сложность метода ветвей и границ, пример Финкельштейна.

11. Понижение показателя степени в верхней оценке числа итераций МВГ за счет предварительной сортировки и рекурсии.

12. Принцип оптимальности Беллмана, табличный вариант метода динамического программирования для задачи о сумме подмножеств в форме распознавания.

13. Табличный вариант метода динамического программирования для задачи о ранце.

14. Метод динамического программирования со списками для задачи о ранце.

15. Концепция ядра и понятие балансировки в методе динамического программирования. Сбалансированные наборы, сбалансированные операции.

16. Алгоритм BALSUB для задачи о сумме подмножеств.

17. Оценки сложности и сравнение различных вариантов метода динамического программирования для задачи о сумме подмножеств.

18. Понятия многогранника, грани, вершины, связь с задачами линейного программирования. Формы задания задачи линейного программирования.

19. Понятие базового допустимого и недопустимого решений задачи ЛП, геометрическая интерпретация.

20. Прямой симплекс-метод, получение начального решения.

21. Двойственный симплекс-метод.

22. Постановка задачи линейного целочисленного программирования. Понятие отсечения. Метод Гомори.

Основная литература

1. Пападимитриу Х., Стайглиц К. Комбинаторная оптимизация. Алгоритмы и сложность. – 1984..

2. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи //Мир. – 1982.Самуэльсон П. Экономикс. М.: Прогресс, 1994

3. Дополнительная учебно-методическая литература

4. Сигал И. Х., Иванова А. П. Введение в прикладное дискретное программирование //М.: физматлит. – 2003.

5. Korte B. et al. Combinatorial optimization. – Heidelberg : Springer, 2012. – Т. 2.

6. Данциг Д. Линейное программирование, его обобщения и применения //М.: Прогресс. – 1966.

7. Kellerer H., Pfershy U., Pisinger D. Knapsack Problems. Springer Verlag, 2004.

8. Martello S., Toth P., Knapsack Problems. John Wiley & Sons Ltd., 1990.

Литература

[1] Kellerer H., Pfershy U., Pisinger D. Knapsack Problems. Springer Verlag, 2004.

[2] Martello S., Toth P., Knapsack Problems. John Wiley & Sons Ltd., 1990.

[3] Сигал И.Х., Иванова А.П. Введение в прикладное дискретное программирование: модели и вычислительные алгоритмы. изд. 2-е, исп. и доп. М.: ФИЗМАТЛИТ, 2007.

[4] Kolesar P.J. A branch and bound algorithm for the knapsack problem // Management Science. 1967. V. 13, N 9. P. 723–735.

[5] Greenberg H., Hegerich R.L. A branch and bound algorithm for the knapsack problem // Management Science. 1970. V. 16, N 5. P. 327-332.

[6] Финкельштейн Ю.Ю. Приближенные методы и прикладные задачи дискретного программирования. М.: Наука, 1976.

[7] Гришухин В.П. Эффективность метода ветвей и границ в задачах с булевыми переменными // Исследования по дискретной оптимизации. 1976. С. 203-230.

[8] Колпаков Р.М., Посыпкин М.А., Асимптотическая оценка сложности метода ветвей и границ с ветвлением по дробной переменной для задачи о ранце. Дискретн. анализ и исслед. опер., 2008, 15:1, 58–81.

[9] Р.М. Колпаков, М.А. Посыпкин, Верхняя и нижняя оценки трудоемкости метода ветвей и границ для задачи о ранце // Дискретная математика, Т. 22, вып. 1, 2010, С. 58-73