Обзор методов разработки алгоритмов

1 Формулировки задач

В этом параграфе сформулируем условия нескольких оптимизационных задач. В каждой из них требуется среди множества возможных вариантов выбрать тот, для которого значение заданной целевой функции достигает экстремального значения (минимального или максимального, в зависимости от условия). Рассматриваемые в пособии методы создания алгоритмов будем применять для решения представленных задач.

1.1 Задача линейного раскроя

Начнем с задачи о линейном раскрое. Пусть даны стержень длиной L и n видов деталей. Деталь вида j имеет длину $l_j > 0$ и стоимость $c_j > 0$, $j = 1, \ldots, n$. Стержень необходимо раскроить на детали таким образом, чтобы их общая стоимость была максимальна.

Построим математическую модель. Для этого введем неизвестные x_j — количество деталей вида j в плане раскроя, $j = 1, \ldots, n$. Тогда общая длина полученных деталей ограничена размером стержня:

$$\sum_{j=1}^{n} l_j x_j \le L. \tag{1}$$

Ограничения на переменные:

$$x_j \ge 0, \quad \text{целые}, \quad j = 1, \ldots, n. \tag{2}$$

Общая стоимость полученных деталей равна:

$$\sum_{j=1}^{n} c_j x_j \longrightarrow \max. \tag{3}$$

Здесь (1), (2) — условия, определяющие множество возможных значений переменных x_j, $j = 1, \ldots, n$. Среди всех возможных решений системы (1), (2) нужно найти то, на котором достигается максимум целевой функции (3).

Рассмотрим пример. Пусть $L = 10$, $n = 2$, $l_1 = 2$, $l_2 = 3$, $c_1 = 4$, $c_2 = 7$. Среди допустимых вариантов раскроя стержня есть такие:

- 5 деталей первого вида ($x_1 = 5$, $x_2 = 0$). Стоимость раскроя равна: $4 \cdot 5 + 7 \cdot 0 = 20$.
- 3 детали второго вида ($x_1 = 0$, $x_2 = 3$). Стоимость: $4 \cdot 0 + 7 \cdot 3 = 21$.
- 2 детали первого вида, 2 детали второго вида ($x_1 = 2$, $x_2 = 2$). Стоимость: $4 \cdot 2 + 7 \cdot 2 = 22$.

Видно, что среди этих трех вариантов третий — самый удачный, так как для него значение целевой функции (стоимость деталей раскроя) максимальна.

В следующих параграфах рассмотрим три различных способа решения задачи линейного раскроя. Первый, наивный, — метод полного перебора. Второй, приближенный, — применение эвристического алгоритма. И третий, точный, — метод динамического программирования.

1.2 Задача о рюкзаке

Рассмотрим задачу о рюкзаке. В литературе часто встречается и другое ее название — задача о ранце. Пусть дан рюкзак грузоподъемностью V и n предметов, каждый из которых имеет вес $a_j > 0$ и стоимость $c_j > 0$, $j = 1, \ldots, n$. Необходимо заполнить рюкзак набором предметов с наибольшей общей стоимостью.

Введем логические переменные x_j — признак наличия предмета j в наборе:

$$x_j = \begin{cases} 1, & \text{если предмет с номером } j \text{ помещен в рюкзак,} \\ 0, & \text{иначе,} \end{cases}$$
$$j = 1, \ldots, n.$$

Общий вес заполненного рюкзака ограничен его грузоподъемностью:

$$\sum_{j=1}^{n} a_j x_j \leq V. \tag{4}$$

Ограничения на переменные:

$$x_j \in \{0, 1\}, \quad j = 1, \ldots, n. \tag{5}$$

Максимизации суммарной стоимости предметов соответствует целевая функция:

$$\sum_{j=1}^{n} c_j x_j \longrightarrow \max. \tag{6}$$

Задача о рюкзаке отличается от задачи о линейном раскрое тем, что в последней число деталей каждого вида (аналог предметов) не ограничено сверху и может быть любым.

Рассмотрим пример. Пусть $V = 10$, $n = 3$, $a_1 = 6$, $a_2 = 5$, $a_3 = 4$, $c_1 = 3$, $c_2 = 4$, $c_3 = 5$. Можно взять первый и третий предметы (общая стоимость 8), а можно второй и третий (стоимость 9). Второй вариант, конечно, лучше.

В пособии представлено три метода решения задачи о рюкзаке: полный перебор, эвристический алгоритм, динамическое программирование.

1.3 Задача о камнях

Дадим формулировку комбинаторной задачи, часто встречающейся в различных приложениях. Пусть имеется n камней с весами $v_1, \ldots, v_n$, $j = 1, \ldots, n$. Необходимо разложить их на m куч так, чтобы вес самой тяжелой кучи был минимальным.

Пусть N_i — множество, элементы которого являются номерами камней, содержащихся в куче i:

$$N_i \subset 1..n, \qquad i = 1, \ldots, m, \tag{7}$$

$$N_i \cap N_k = \emptyset, \qquad i \neq k. \tag{8}$$

Для удобства введем переменные x_j — номер кучи, в которую помещен камень j:

$$x_j = i, \qquad \text{если } j \in N_i. \tag{9}$$

Для решения задачи нужно найти минимум функции:

$$\max_{i=1,\ldots,m} \left\{ \sum_{j \in N_i} v_j \right\} \longrightarrow \min. \tag{10}$$

Для примера, пусть $m = 2$, $n = 5$, веса камней: $v_1 = 2$, $v_2 = 3$, $v_3 = 5$, $v_4 = 8$, $v_5 = 9$. Искомое разбиение на две кучи следующее. Первая куча содержит камни с весами 2, 3 и 9, вторая — 5 и 8. Вес самой тяжелой

кучи (первой) равен 14. Значения переменных в оптимальном решении таковы:

$$x_1 = 1, \quad x_2 = 1, \quad x_3 = 2, \quad x_4 = 2, \quad x_5 = 1.$$

Множества N_1 и N_2 равны:

$$N_1 = \{1, 2, 5\}, \qquad N_1 = \{3, 4\}.$$

Заметим, что переменные x_j с индексами из множества N_1 равны 1, а с индексами из N_2 равны 2, что согласуется с введенными выше определениями.

В данном пособии задача о камнях решается тремя способами: полный перебор, применение эвристики, метод локального поиска.

1.4 Задача коммивояжера

Рассмотрим задачу коммивояжера (о бродячем торговце). Именно на этой задаче в первую очередь проверяется работоспособность многих методов решения трудных задач (NP-полных, подробнее о них см. в [5]).

Имеется n городов с номерами $1, 2, \ldots, n$, для каждой пары городов i и j задано расстояние $c[i, j]$ между ними. Выезжая из города 1, коммивояжер должен побывать во всех остальных городах по одному разу и вернуться в исходный город 1. Определить, в каком порядке следует объезжать города, чтобы суммарное пройденное расстояние было наименьшим.

Пусть $1, p_1, p_2, \ldots, p_{n-1}, 1$ — номера городов, записанные в порядке их обхода. То есть p_i — номер города, посещаемого на i-м шаге, $i = 0, \ldots, n$, $p_0 = p_n = 1$. Тогда пройденное расстояние равно:

$$\sum_{j=0}^{n-1} c[p_j, p_{j+1}] \longrightarrow \min. \tag{11}$$

Среди чисел $p_1, p_2, \ldots, p_{n-1}$ ровно по одному разу встречается каждое число из интервала $2..n$. Таким образом, в задаче ищется перестановка целых чисел от 2 до n, доставляющая минимум целевой функции (11).

Рассмотрим пример. Пусть $n = 5$, матрица расстояний между горо-

дами:

$$C = \begin{pmatrix} 0 & 5 & 3 & 1 & 2 \\ 5 & 0 & 1 & 3 & 6 \\ 3 & 1 & 0 & 4 & 2 \\ 1 & 3 & 4 & 0 & 7 \\ 2 & 6 & 2 & 7 & 0 \end{pmatrix}.$$

Оптимальный маршрут выглядит так:

$$1 \to 4 \to 2 \to 3 \to 5 \to 1.$$

Пройденное коммивояжером расстояние равно 9.

Задачу коммивояжера можно решать различными способами. В данном пособии применяются методы полного перебора и локального поиска.

1.5 NP-полнота

В этом разделе очень коротко приведем некоторые сведения из теории сложности. Для более подробного ознакомления рекомендуем [5].

Как правило, с задачей связан некоторый параметр n. Например, в задаче о рюкзаке n — это число предметов, в задаче коммивояжера n — число городов. Часто от этого параметра зависит время работы $T(n)$ алгоритма решения задачи.

Напомним ряд определений. Полином (многочлен) $P_k(x)$ степени k есть выражение:

$$P_k(x) = c_k x^k + c_{k-1} x^{k-1} + \ldots + c_1 x + c_0,$$

где c_j — постоянные коэффициенты, $j = 0, \ldots, k$.

Будем говорить, что $f(x) = O(g(x))$, если существует константа $c > 0$ и x_0, что для всех $x > x_0$ выполнено:

$$f(x) \leq c \cdot g(x).$$

Говорят, что $g(x)$ — оценка сверху функции $f(x)$. Аналогично можно ввести и нижнюю оценку. Будем считать, что $f(x) = \Omega(g(x))$, если существует константа $c > 0$ и x_0, что для всех $x > x_0$ выполнено:

$$f(x) \geq c \cdot g(x).$$

Алгоритм называется полиномиальным, если время его работы $T(n)$ ограничено сверху некоторым полиномом:

$$T(n) = O(P_k(n)).$$

Алгоритм называется экспоненциальным, если время его работы $T(n)$ ограничено снизу показательной функцией:

$$T(n) = \Omega(a^n),$$

где $a > 1$.

В теории сложности рассматриваются только задачи разрешимости. У задачи разрешимости может быть всего два возможных ответа: "да" или "нет". Приведем пример такой задачи.

Пусть дано n городов $1, 2, \ldots, n$ и известны расстояния между всеми парами городов. Выезжая из города 1, коммивояжер должен побывать во всех остальных городах по одному разу и вернуться в исходный город. Определить, существует ли такой порядок обхода городов, при котором пройденное расстояние не больше чем l.

Задача в такой формулировке напоминает рассмотренную нами ранее оптимизационную задачу коммивояжера. Отличие между ними состоит в том, что в одной ищется маршрут наименьшей длины, а в другой необходимо дать ответ на вопрос о существовании маршрута с не более чем заданной длиной. Заметим, что, решив задачу в оптимизационной постановке, можно дать ответ на вопрос задачи разрешимости. Действительно, для этого достаточно сравнить длину кратчайшего маршрута с l.

Дадим теперь определения сложностных классов задач P и NP (напомним, сейчас мы говорим только о задачах разрешимости). Класс P состоит из задач, для которых существуют полиномиальные алгоритмы решения. Класс NP составляют задачи, для которых существуют полиномиальные алгоритмы проверки правильности предъявляемого решения (точнее, если ответ задачи "да", то существует некоторая подсказка, позволяющая за полиномиальное время получить этот ответ). Неформально говоря, класс P состоит из задач, которые можно быстро решить, а класс NP — из задач, решение которых можно быстро проверить. Эти определения не вполне строгие, более точно они даны в [5].

Вот пример задачи класса P. Требуется определить, есть ли в числовом массиве $A[1..n]$ элемент со значением не меньшим, чем k. Очевидный

в данном случае алгоритм решения перебирает все элементы массива за время $O(n)$.

Примером задачи класса NP служит задача коммивояжера в постановке задачи разрешимости. Действительно, если нам *дан* некоторый маршрут длиной не более чем k (неважно, откуда он взялся или как мы его получили), то за время $O(n)$ можно проверить, что он действительно имеет такую длину, и тем самым убедиться в его существовании. Для этого нужно всего лишь перебрать все n содержащихся в нем переходов между городами и просуммировать их длины.

Очевидно включение $\mathrm{P} \subset \mathrm{NP}$ (для проверки решения задачи класса P достаточно решить ее полиномиальным алгоритмом). Вопрос, является ли это включение строгим, остается одним из центральных вопросов теории сложности. Большинство специалистов полагают, что $\mathrm{P} \neq \mathrm{NP}$. Одним из аргументов в пользу последнего служит существование NP-полных задач.

Задача называется NP-полной, если она принадлежит классу NP и к ней за полиномиальное время можно свести любую другую задачу этого класса. Если какая-нибудь (любая) NP-полная задача имеет полиномиальный алгоритм решения, то все NP-полные задачи полиномиально разрешимы и, как следствие, P=NP (в этом случае за полиномиальное время разрешимы все задачи класса NP). Можно сказать и иначе. Если P=NP, то NP-полные задачи разрешимы за полиномиальное время.

NP-полные задачи являются наиболее трудными в классе NP. Всем сформулированным в этом разделе задачам соответствуют NP-полные задачи разрешимости. Ни для одной из них никому не удалось построить полиномиальный алгоритм решения. Это укрепляет нас во мнении, что $\mathrm{P} \neq \mathrm{NP}$.

В этом пособии речь идет в основном о задачах оптимизации. Однако, как мы уже видели, для задачи оптимизации можно построить соответствующую ей задачу разрешимости. При этом решение задачи разрешимости приведет нас к решению задачи оптимизации. Таким образом, задачи оптимизации в определенном смысле проще задач разрешимости. И если для задачи разрешимости будет каким-то образом доказана ее сложность (с точки зрения времени решения), то и соответствующая ей задача оптимизации также сложна.

Позволим себе некоторую вольность — говоря о NP-полноте задачи оптимизации мы будем иметь ввиду NP-полноту соответствующей ей задачи разрешимости.

Для чего нужны эти определения? Дело в том, что, как правило, полиномиальные алгоритмы работают относительно быстро, а алгоритмы со сложностью, превышающей полиномиальную, — медленно. Эти утверждения не следует воспринимать как догму. Трудно назвать быстрым алгоритм со временем работы

$$T(n) = 10^{100}n \quad \text{или} \quad T(n) = n^{100}.$$

Иногда вместо полиномиальных применяют неполиномиальные алгоритмы решения. Например, задачи линейного программирования относятся к классу полиномиальных задач, однако на практике эффективнее других зарекомендовал себя симплексный метод решения, имеющий в худшем случае экспоненциальную сложность [12].

Но, в общем случае, для решения задачи стремятся построить именно полиномиальный алгоритм. Если же установлена NP-полнота задачи (например, методом сведения к другой NP-полной задаче), то построить полиномиальный алгоритм вряд ли удастся и нужно искать *другие* методы решения, например использование эвристик.

Так как многие рассматриваемые в этом пособии задачи NP-полные, то для них не найдены алгоритмы решения с полиномиальной временной сложностью. Именно поэтому для каждой из задач предлагается несколько способов решения.

2 Переборные методы

Начнем с описания методов, основанных на полном переборе всех возможных решений задачи. Для всех представленных выше задач множество возможных решений конечно и, перебрав все из них, мы можем найти лучшее. Как правило, полный перебор невозможно осуществить на практике из-за огромного числа рассматриваемых вариантов. Однако этот метод может применяться в случае малых размеров входных данных или в виде частичного перебора в других алгоритмах.

Таким образом, для реализации полного перебора нужна процедура генерации всех допустимых решений. Как правило, речь идет о переборе всех перестановок целых чисел из некоторого диапазона, генерации всех подмножеств конечного множества и т. д.

Мы не будем рассматривать методы генерации комбинаторных объектов, прекрасный обзор таких алгоритмов содержится в [6].

Для оценивания времени работы алгоритмов, основанных на полном переборе, подсчитаем число возможных решений для некоторых задач.

2.1 Задача о рюкзаке

Каждый предмет может быть либо помещен в рюкзак, либо нет. Соответственно каждая переменная x_j может принимать одно из двух возможных значений — 1 или 0. Любому решению соответствует вектор из n нулей и единиц. Всего таких векторов 2^n.

Рассмотрим пример п. 1.2. Все возможные комбинации значений переменных x_1, x_2, x_3 и соответствующие им наборы предметов выпишем в таблицу:

x_1	x_2	x_3	набор	вес	стоимость
0	0	0	$\emptyset$	0	**0**
1	0	0	$\{1\}$	6	**3**
0	1	0	$\{2\}$	5	**4**
0	0	1	$\{3\}$	4	**5**
1	1	0	$\{1, 2\}$	11	7
1	0	1	$\{1, 3\}$	10	**8**
0	1	1	$\{2, 3\}$	9	**9**
1	1	1	$\{1, 2, 3\}$	15	12

Жирным выделены стоимости наборов, помещающихся в рюкзак. Среди них выбираем набор с наибольшей стоимостью — в него входят второй и третий предметы.

2.2 Задача о камнях

Для решешия задачи о камнях методом полного перебора нужно перебрать все возможные разбиения множества камней на m подмножеств (куч):

x_1	x_2	x_3	x_4	x_5	разбиение	наиб. вес кучи
1	1	1	1	1	{1, 2, 3, 4, 5}	27
1	1	1	1	2	{1, 2, 3, 4}, {5}	18
1	1	1	2	1	{1, 2, 3, 5}, {4}	19
1	1	1	2	2	{1, 2, 3}, {4, 5}	17
1	1	2	1	1	{1, 2, 4, 5}, {3}	22
1	**1**	**2**	**1**	**2**	**{1, 2, 4}, {3, 5}**	**14**
1	**1**	**2**	**2**	**1**	**{1, 2, 5}, {3, 4}**	**14**
1	1	2	2	2	{1, 2}, {3, 4, 5}	22
1	2	1	1	1	{1, 3, 4, 5}, {2}	24
1	2	1	1	2	{1, 3, 4}, {2, 5}	15
1	2	1	2	1	{1, 3, 5}, {2, 4}	16
1	2	1	2	2	{1, 3}, {2, 4, 5}	20
1	2	2	1	1	{1, 4, 5}, {2, 3}	19
1	2	2	1	2	{1, 4}, {2, 3, 5}	17
1	2	2	2	1	{1, 5}, {2, 3, 4}	16
1	2	2	2	2	{1}, {2, 3, 4, 5}	25

Здесь рассматривается пример п. 1.3 (две кучи, пять камней с весами 2, 3, 5, 8, 9). Для определенности куче, содержащей первый камень, присваивается номер 1, другой — номер 2. Жирным шрифтом выделены два варианта оптимальных разбиений.

2.3 Задача коммивояжера

В задаче коммивояжера нужно найти перестановку городов $A_1, \ldots, A_n$, для которой пройденное расстояние минимально. Заметим, что первый элемент искомой перестановки всегда равен 1. Поэтому при полном переборе переставлять нужно оставшиеся $n-1$ города. Всего таких перестановок $(n-1)!$.

При $n = 5$ получаем $4! = 24$ различных перестановок. Если при этом матрица задачи симметрическая, то получаем 12 перестановок (см. зада-

чу 2). Для примера п. 1.4 все возможные маршруты выписаны в таблицу:

маршрут	стоимость
$1 \to 2 \to 3 \to 4 \to 5 \to 1$	19
$1 \to 2 \to 3 \to 5 \to 4 \to 1$	16
$1 \to 2 \to 4 \to 3 \to 5 \to 1$	16
$1 \to 2 \to 4 \to 5 \to 3 \to 1$	20
$1 \to 2 \to 5 \to 3 \to 4 \to 1$	18
$1 \to 2 \to 5 \to 4 \to 3 \to 1$	25
$1 \to 3 \to 2 \to 4 \to 5 \to 1$	16
$1 \to 3 \to 2 \to 5 \to 4 \to 1$	18
$1 \to 3 \to 4 \to 2 \to 5 \to 1$	18
$1 \to 3 \to 5 \to 2 \to 4 \to 1$	15
$\mathbf{1 \to 4 \to 2 \to 3 \to 5 \to 1}$	**9**
$1 \to 4 \to 3 \to 2 \to 5 \to 1$	14

Жирным шрифтом выделен маршрут с наименьшей стоимостью.

2.4 Метод ветвей и границ

Если допустимое множество решений задачи конечно, то метод полного перебора всегда приводит к оптимальному решению. Однако число допустимых решений растет очень быстро с ростом параметров задачи. Поэтому этот метод крайне неэффективен. Одним из вариантов ускорения процесса является отбрасывание подмножеств заведомо неоптимальных решений. Эта идея улучшенного перебора применяется в методе ветвей и границ. Опишем его общую схему. Пусть решается задача оптимизации:

$$\varphi(x) \longrightarrow \min, \tag{12}$$

$$x \in \Omega. \tag{13}$$

Определение. Оценкой снизу функции φ на подмножествах множества Ω называется функция $\psi : 2^\Omega \to R$ (2^Ω — множество всех подмножеств множества Ω), для которой:

$$\psi(A) \le \min_{x \in A} \varphi(x), \qquad \text{для любого } A \subset \Omega. \tag{14}$$

Рассмотрим разбиение множества Ω на k непустых подмножеств:

$$\begin{array}{l} \Omega = \Omega_1 \cup \Omega_2 \cup \ldots \Omega_k, \\ \Omega_i \neq \emptyset, \quad i = 1, \ldots, k, \\ \Omega_i \cap \Omega_j = \emptyset, \quad i \neq j. \end{array} \tag{15}$$

Предположим, что известно некоторое допустимое решение x_0, выбранное случайно или полученное применением какого-либо эвристического алгоритма. Примем начальное рекордное решение x_{rec} равным x_0.

Введем два семейства подмножеств: A и B. В семейство A включим те подмножества Ω_i разбиения (15), для которых оценка $\psi(\Omega_i)<\varphi(x_{rec})$. Остальные подмножества поместим в семейство B. Таким образом, разбиение (15) множества Ω состоит из двух непересекающихся частей A и B.

Заметим, что подмножества семейства B не могут содержать оптимального решения задачи (12)–(14) (Почему?).

Опишем основную итерацию алгоритма.

Выберем и удалим из семейства A подмножество Ω_s с минимальной оценкой.

Если возможно быстро найти минимум функции $\varphi(x)$ на подмножестве Ω_s, то сделаем это. Пусть x_s — полученное решение. Если $\varphi(x_s) < \varphi(x_{rec})$, то запомним новый рекорд: $x_{rec} = x_s$.

Если задача поиска минимума на подмножестве Ω_s вычислительно трудна, то построим разбиение Ω_s на l подмножеств:

$$\Omega_s = \Omega_{s1} \cup \Omega_{s2} \cup \ldots \Omega_{sl}. \tag{16}$$

Подмножества разбиения (16) с оценкой, не меньшей, чем $\varphi(x_{rec})$, поместим в семейство B. Остальными подмножествами заменим подмножество Ω_s в семействе A.

Если семейство A непустое, то выполняем основную итерацию еще раз.

Конечно, это весьма общая схема и требует уточнения. Необходимо:

1. Выбрать способ оценивания подмножеств. С одной стороны, оценка должна быть как можно ближе к минимуму функции на подмножестве, с другой — должна легко вычисляться.

2. Определить способ разбиений множеств на подмножества.

3. Уточнить порядок выбора очередного подмножества для ветвления. Он может отличаться от предложенного выше. Например, можно выбирать из подмножеств, полученных на последнем шаге, подмножество с лучшей оценкой (а не из всех).

Название метода объясняется следующим. “Ветвей” — означает, что задача “ветвится” на подзадачи (каждому подмножеству разбиения соответствует задача поиска минимума функции $\varphi(x)$ на этом подмножестве), “границ” — подмножества разбиения оцениваются и отсекаются те из них, для которых оценка не меньше текущего рекордного значения. В худшем случае метод ветвей и границ сводится к полному перебору всех возможных вариантов решения задачи.

2.5 Задачи

В задачах 3–8 этого раздела требуется написание алгоритмов перебора различных комбинаторных объектов. Такие алгоритмы могут использоваться в качестве вспомогательных при реализации методов решения, основанных на полном переборе.

1. Подсчитайте, сколько существует допустимых вариантов решения задачи о линейном раскрое.

2. Сколько будет вариантов возможных решений в задаче коммивояжера, если матрица C — симметрическая?

3. Составьте процедуру генерации всех перестановок целых чисел от 1 до n.

4. Предложите алгоритм построения всех способов представления целого числа n в виде суммы $n = A_1 + A_2 + \ldots + A_k$, где k: а) зафиксировано, б) произвольно, в) фиксировано и числа в сумме не повторяются.

5. Напишите алгоритм генерации всех разбиений n элементного множества на k подмножеств, где k: а) зафиксировано, б) произвольно.

6. Придумайте алгоритм генерации всех счастливых билетов из $2n$ цифр в k-значной системе счисления (сумма первых n цифр билета равна сумме n последних).

7. Разработайте алгоритм генерации всех m-элементных подмножеств множества целых 1 до n.

8. Разработайте алгоритм генерации всех подмножеств заданного n-элементного множества.

3 Жадные алгоритмы

Для решения многих задач есть простые и быстрые алгоритмы, например те, которые называются жадными. Жадный алгоритм делает на каждом шаге локально оптимальный выбор и в дальнейшем этот выбор не отменяется. Рассмотрим применение жадного алгоритма для решения задачи о выборе заявок [5]. Это единственная в данном пособии задача, для которой мы предлагаем единственный метод решения.

3.1 Задача о выборе заявок

Пусть даны n заявок на проведение занятий в одной и той же аудитории. Два различных занятия не могут перекрываться по времени. В каждой заявке указаны начало и конец занятия. Разные заявки могут пересекаться, и тогда можно удовлетворить только одну из них. Необходимо набрать максимальное количество совместимых друг с другом заявок.

Пусть $\alpha_1, \ldots, \alpha_n$ — моменты начала, а $\beta_1, \ldots, \beta_n$ — моменты окончания занятий с номерами от 1 до n. Тогда $[\alpha_i,\ \beta_i)$ — промежуток времени проведения занятия с номером i. Необходимо выборать максимальное число занятий так, чтобы соответствующие им промежутки не пересекались (моменты начала одного занятия могут совпадать с моментами начала следующего за ним).

Будем считать, что занятия упорядочены по времени их окончания:

$$\beta_1 \leq \ldots \leq \beta_n.$$

Идея алгоритма такова. Выберем первую заявку (для нее будет минимальным время окончания занятия). Затем просматриваем по порядку все заявки, начиная со второй, и берем те, у которых время начала не меньше, чем время окончания последней выбранной заявки.

Запишем то же самое с помощью псевдокода:

print 1
$j = 1$
for $i = 2$ to n do
    if $\alpha_i \geq \beta_j$ then
        print i
        $j = i$

В начале работы выводится на печать номер первой выбранной заявки — 1. Переменная j хранит номер последней выбранной заявки и сначала j равно 1. Затем в цикле перебираются все заявки с 2 по n. Если время начала занятия заявки i не меньше, чем время окончания у заявки j, то заявка i принимается и переменная j запоминает ее номер. Отметим, что предложенный алгоритм имеет полиномиальную временную сложность работы.

Задание. Докажите правильность работы этого алгоритма.

Рассмотрим пример. Пусть дано n заявок с моментами начала и окончания занятий, указанных в таблице (заявки упорядочены по времени окончания, исчисление времени условное):

заявка	время начала	время окончания
a	*2*	*4*
b	*1*	*5*
c	*4*	*8*
d	*3*	*10*
e	*9*	*10*
f	*5*	*12*
g	*7*	*14*
h	*13*	*15*

Следуя предложенному алгоритму, выберем первую заявку — **a**. Время ее окончания равно *4*. Заявки, у которых время начала не менее *4*, следующие:

$$\mathbf{c}, \quad \mathbf{e}, \quad \mathbf{f}, \quad \mathbf{g}, \quad \mathbf{h}.$$

Берем первую из них — заявку **c**. Заявки, со временем начала больше или равным времени окончания заявки **c**:

$$\mathbf{e}, \quad \mathbf{h}.$$

Первая среди них — заявка **e**, ее время окончания равно *10*. Теперь на выбор остается только одна заявка:

$$\mathbf{h},$$

которую мы и берем. Таким образом, мы взяли четыре заявки:

$$\mathbf{a}, \quad \mathbf{c}, \quad \mathbf{e}, \quad \mathbf{h}. \tag{17}$$

Отметим свойства задач, для решения которых применимы жадные алгоритмы:

1. Свойство оптимальности для подзадач — оптимальное решение задачи содержит оптимальные решения подзадач.

2. Свойство жадного выбора — последовательность локально оптимальных выборов приводит к оптимальному решению.

Поясним эти свойства. Многие задачи можно рассматривать на некотором "сужении" множества допустимых решений. Например, задачу о выборе заявок можно рассмотреть только для тех заявок, время окончания выполнения которых не превосходит заданного t. Задачу о линейном раскрое стержня длиной L можно рассмотреть для раскроя стержня некоторой длины y ($0 \leq y \leq L$). Это и есть подзадачи. Вернемся к нашему примеру. Ограничим задачу множеством заявок со временем окончания, меньшим или равным *10*:

$$\mathbf{a}, \quad \mathbf{b}, \quad \mathbf{c}, \quad \mathbf{d}, \quad \mathbf{e}.$$

Оптимальное решение для этой подзадачи содержит три заявки:

$$\mathbf{a}, \quad \mathbf{c}, \quad \mathbf{e}.$$

Заметим, что оптимальное решение (17) исходной задачи содержит тоже три заявки со временем окончания, не большим *10*. В данном случае это те же три заявки. Обобщив, мы можем сказать, что оптимальное решение (17) содержит оптимальные решения всех подзадач.

Обсудим теперь второе свойство (жадного выбора). На каждом шаге рассмотренного алгоритма мы придерживались определенной стратегии — выбирали среди подходящих заявок ту, у которой время окончания занятия наименьшее. Можно доказать, что именно *такой* способ увеличения числа взятых заявок на каждом шаге приводит к оптимальному решению. То есть при таких "жадных" выборах не отсекаются возможные пути к оптимальному решению.

Жадные алгоритмы применяются для решения многих задач, например, для решения задачи построения минимального покрывающего дерева [2]. Жадные алгоритмы могут быть основой различных эвристических алгоритмов, в том числе алгоритмов решения задач линейного раскроя, о рюкзаке, о камнях, коммивояжера.

3.2 Эвристические методы

Далее в этом разделе рассмотрим алгоритмы, дающие приближенные (неточные) решения задач. В основе таких эвристических алгоритмов лежат идеи, основанные на интуиции, жизненном опыте. Для задач оптимизации они могут давать далеко не оптимальные решения. Поэтому, если нужно найти точное решение задачи, эти методы неприменимы. В то же время их просто понять, легко запрограммировать. Эвристические алгоритмы обычно имеют малую сложность вычислений и часто выдают хорошие, близкие к оптимальным, решения. Поэтому они часто применяются для решения трудных задач.

Рассмотрим применение приближенных алгоритмов для решения нескольких задач. В основе этих алгоритмов будут лежать "жадные" эвристики.

3.3 Задача линейного раскроя

Пусть выполнено:

$$\frac{c_1}{l_1} \geq \frac{c_2}{l_2} \geq \ldots \geq \frac{c_n}{l_n}, \tag{18}$$

то есть все детали упорядочены по стоимости единицы длины.

В качестве эвристики применим следующий жадный алгоритм. Раскроим стержень на максимальное количество деталей первого вида:

$$x_1 = \left\lfloor \frac{L}{l_1} \right\rfloor, \quad y_1 = L \bmod l_1, \tag{19}$$

где y_1 — длина остатка материала. Остаток стержня можно раскроить на детали второго вида, следующий остаток стержня — на детали третьего вида и т. д.:

$$x_j = \left\lfloor \frac{y_{j-1}}{l_j} \right\rfloor, \quad y_j = y_{j-1} \bmod l_j, \quad j = 2, \ldots, n. \tag{20}$$

Рассмотрим пример п. 1.1. Стержень длиной 10 необходимо раскроить на детали длиной 2 и 3, стоимостью соответственно 4 и 7. Упорядочим детали в соответствии с (18):

$$\frac{7}{3} \geq \frac{4}{2}\,.$$

Следуя алгоритму, раскроим стержень на детали длиной 3. Всего их получится три штуки. Остатка стержня, равного 1, не хватит для изготовления деталей другого вида.

Заметим, что для этого примера предложенный эвристический алгоритм не дал оптимального решения.

3.4 Задача о рюкзаке

Пусть предметы упорядочены по отношению стоимости к весу:

$$\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \ldots \geq \frac{c_n}{a_n}. \tag{21}$$

Рассмотрим следующий жадный алгоритм. Вначале набор предметов пуст. Будем последовательно перебирать предметы с номерами $1, 2, \ldots, n$. Очередной предмет помещаем в рюкзак в том случае, если его добавление в набор не приводит к превышению грузоподъемности рюкзака.

Формально

$$x_1 = \begin{cases} 1, & \text{если } a_1 \leq V, \\ 0, & \text{иначе.} \end{cases} \tag{22}$$

Остаток грузоподъемности рюкзака:

$$y_1 = V - a_1 x_1. \tag{23}$$

Остальные переменные определяются аналогично:

$$\begin{gathered} x_j = \begin{cases} 1, & \text{если } a_j \leq y_{j-1}, \\ 0, & \text{иначе,} \end{cases} \\ y_j = y_{j-1} - a_j x_j, \\ j = 2, \ldots, n. \end{gathered} \tag{24}$$

Рассмотрим пример п. 1.2. Рюкзак грузоподъемностью 10 необходимо заполнить предметами с весами 6, 5 и 4, стоимостями соответственно 3, 4 и 5. Упорядочим детали в соответствии с (21):

$$\frac{5}{4} \geq \frac{4}{5} \geq \frac{3}{6}.$$

Следуя алгоритму, возьмем сначала предмет весом 4, а затем весом 5.

Заметим, что, применив эвристический алгоритм для этого примера, мы получили оптимальное решение.

Задание. Приведите пример, для которого эвристический алгоритм дает не лучшее решение.

3.5 Задача о камнях

Предлагается следующий жадный алгоритм. На каждом шаге будем брать самый тяжелый камень из оставшихся и класть его в самую легкую кучу. Пусть камни упорядочены по убыванию весов:

$$v_1 \geq v_2 \geq \ldots \geq v_n. \tag{25}$$

Обозначим b_i — вес кучи i, $i = 1, \ldots, m$. Приведем псевдокод алгоритма:

for $i = 1$ to m do
    b_i=0
for $j = 1$ to n do
    $k = \underset{i=1,\ldots,m}{\operatorname{argmin}}\{b_i\}$
    $x_j = k$
    $b_k = b_k + v_j$

3.6 Задачи

1. Задача о выборе аудиторий. Пусть даны n заявок на проведение занятий и неограниченное множество аудиторий. Два различных занятия не могут перекрываться по времени в одной и той же аудитории. В каждой заявке указаны начало и конец занятия. Разные занятия могут пересекаться, и тогда они должны проходить в разных аудиториях. Необходимо распределить заявки по аудиториям, используя как можно меньше аудиторий.

2. Задача о выборе бензоколонок. Профессор едет по шоссе из Петербурга в Москву, имея при себе карту с указанием всех стоящих на шоссе бензоколонок и расстояний между ними. Известно расстояние, которое может проехать машина с полностью заправленным баком. Разработайте эффективный алгоритм, позволяющий выяснить, на каких бензоколонках надо заправляться, чтобы количество заправок было минимально. В начале пути бак полон.

3. Предложите эвристический алгоритм решения задачи коммивояжера.

4. Составьте контрпример для эвристического алгоритма задачи о камнях (пример, на котором алгоритм дает неоптимальное решение).

5. Организации нужно нанять переводчиков для перевода с определенного множества языков. Каждый из имеющихся переводчиков владеет некоторыми иностранными языками и требует определенную зарплату. Требуется определить, каких переводчиков следует нанять, чтобы сумма расходов на зарплату была минимальной.

6. Поставщику нужно доставить товары своим потребителям. Имеется множество возможных маршрутов, каждый из которых позволяет обслужить определенное подмножество потребителей и требует определенных расходов. Требуется определить, какие маршруты следует использовать, чтобы все потребители были обслужены, а сумма транспортных расходов была минимальной.

7. Напишите алгоритм раскраски вершин произвольного графа минимальным числом цветов.

4 Динамическое программирование

Иногда перебор допустимых решений в ходе решения задачи удается существенно сократить применением методов динамического программирования. Динамическое программирование — это процесс пошагового решения задач, когда на каждом шаге из множества допустимых решений выбирается одно решение, оптимизирующее целевую функцию. В методе ветвей и границ задача делится на непересекающиеся подзадачи, решением исходной задачи является лучшее из решений подзадач. Динамическое программирование применяют тогда, когда у задачи есть одинаковые подзадачи. При этом каждая из подзадач решается только один раз и ее решение запоминается в специальной таблице. Это позволяет не искать решения одних и тех же подзадач многократно, при необходимости ответ берется из таблицы.

В основе алгоритма, основанного на динамическом программировании, обычно лежит рекуррентное соотношение, связывающее оптимальные значения целевых функций подзадач. Рассмотрим применения этого метода для решения различных задач.

4.1 Задача линейного раскроя

В случае, когда параметры L, l_j — целые числа, $j = 1, \ldots, n$, метод динамического программирования можно применить для решения задачи линейного раскроя.

Введем обозначение:

$$\varphi(y) = \max \left\{ \sum_{j=1}^{n} c_j x_j \;\middle|\; \sum_{j=1}^{n} l_j x_j \le y,\ x_j \ge 0,\ \text{целое},\ j = 1, \ldots, n \right\}. \quad (26)$$

То есть $\varphi(y)$ — стоимость оптимального раскроя стержня длиной y. Нас интересует $\varphi(L)$.

Покажем, как можно вычислить значения $\varphi(y)$ при известных $\varphi(y')$, $y' < y$.

Пусть l_0 равно минимальной длине детали:

$$l_0 = \min_{j=1,\ldots,n} l_j.$$

Очевидно, что

$$\varphi(y) = 0, \quad y = 0, \ldots, l_0 - 1, \quad (27)$$

так как стержни с длинами, меньшими длин любой из деталей, раскроить не удастся.

Для вычисления значений $\varphi(y)$ при $y \ge l_0$ можно воспользоваться следующим *рекуррентным соотношением Беллмана*:

$$\varphi(y) = \max_{j=1,\ldots,n} \{\varphi(y - l_j) + c_j \mid l_j \le y\}, \quad y = l_0, \ldots, L. \quad (28)$$

Справедливость формулы (28) можно доказать, используя метод математической индукции (проделайте это). На основе соотношения (28) легко составить алгоритм решения задачи линейного раскроя. Обращаем внимание, что для хранения значений функции $\varphi(y)$ мы будем использовать массив φ (доступ к элементам массива обозначается $\varphi[y]$). Приведем псевдокод этого алгоритма:

```
for y = 0 to L do
    φ[y]=0
    ψ[y]=0
    for j = 1 to n do
        if (l_j ≤ y) and (φ[y] < φ[y − l_j] + c_j) then
            φ[y] = φ[y − l_j] + c_j
            ψ[y] = j
```

В цикле по y последовательно перебираются длины стержня от 0 до L. Для каждого y для расчета соответствующего значения $\varphi[y]$ применяется соотношение (28).

Массив ψ применяется для хранения номеров деталей, доставляющих максимум значениям $\varphi[y]$. То есть, если максимум $\varphi[y]$ достигается при $j = k$, то $\psi[y] = k$. Это означает, что от куска длиной y нужно отрезать деталь с номером k (после этого останется кусок длиной $y - l_k$, номер следующей отрезаемой детали равен $\psi[y - l_k]$ и т. д.).

Для вычисления значений переменных x_j можно воспользоваться следующей циклической процедурой:

for $j = 1$ to n do
    $x_j = 0$
$y = L$
while $\psi[y] > 0$ do
    $k = \psi[y]$
    $x_k = x_k + 1$
    $y = y - l_k$

Временная сложность алгоритма решения задачи равна $O(L \cdot n)$ (в основной программе два вложенных цикла). Отметим, что существуют и более быстрые алгоритмы, основанные на соотношении (28), например метод скачков [9].

Рассмотрим применение этого алгоритма на примере. Пусть $L = 10$, $n = 2$, $l_1 = 2$, $l_2 = 3$, $c_1 = 4$, $c_2 = 7$.
Так как наименьшая длина детали равна 2, то

$$\varphi[0] = \varphi[1] = 0,$$

$$\psi[0] = \psi[1] = 0.$$

Для отрезка стержня, по длине равного первой детали, имеем:

$$\varphi[l_1] = c_1,$$

то есть

$$\varphi[2] = 4$$

и

$$\psi[2] = 1.$$

Далее, для $y = 3$:

$$\varphi[3] = \max\{\varphi[3 - l_1] + c_1,\ \varphi[3 - l_2] + c_2\}$$

и получаем:

$$\varphi[3] = \max\{\varphi[3 - 2] + 4,\ \varphi[3 - 3] + 7\} = \max\{\varphi[1] + 4,\ \varphi[0] + 7\} =$$

$$= \max\{0 + 4,\ 0 + 7\} = 7.$$

Так как максимум достигается на втором элементе, то

$$\psi[3] = 2.$$

При $y = 4$ имеем:

$$\varphi[4] = \max\{\varphi[4 - l_1] + c_1,\ \varphi[4 - l_2] + c_2\}$$

$$= \max\{\varphi[2] + 4,\ \varphi[1] + 7\} = \max\{4 + 4,\ 0 + 7\} = 8.$$

Максимум достигается на первом элементе, поэтому

$$\psi[4] = 1.$$

Далее аналогично определяются значения $\varphi[5], \ldots, \varphi[10]$ и $\psi[5], \ldots, \psi[10]$. Полученные значения запишем в таблицу:

y	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
$\varphi[y]$	0	0	4	7	8	11	14	15	18	21	22
$\psi[y]$	0	0	1	2	1	1	2	1	1	2	1

Стоимость оптимального раскроя стержня равна $\varphi[10] = 22$. По значению $\psi[10] = 1$ определяем номер отрезаемой от стержня детали. Отрезав один раз первую деталь, получим остаток стержня длиной 8. Так как $\psi[8] = 1$, то первую деталь нужно отрезать еще раз. Получаем кусок длиной 6. Значение $\psi[6] = 2$, следовательно, теперь нужно отрезать вторую деталь. У нас остается остаток длиной 3. Видим, что $\psi[3] = 2$ — это вторая деталь, после этого $\psi[0] = 0$ и процесс резки завершен. В данном примере остатка стержня не остается. Обе детали отрезались по два раза, поэтому $x_1 = 2$ и $x_2 = 2$.

4.2 Задача о рюкзаке

Применим динамическое программирование для решения задачи о рюкзаке. Будем предполагать, что V, a_j — целые числа, $j = 1, \ldots, n$.

Введем обозначение:

$$\varphi_k(y) = \max \left\{ \sum_{j=1}^{k} c_j x_j \;\middle|\; \sum_{j=1}^{k} a_j x_j \le y,\ x_j \in \{0,1\},\ j = 1, \ldots, n \right\}. \quad (29)$$

То есть $\varphi_k(y)$ — максимальная стоимость набора среди первых k предметов, помещенного в ранец грузоподъемностью y. Нас интересует $\varphi_n(V)$.

По аналогии с задачей линейного раскроя значения $\varphi_k(y)$ можно вычислить, используя *рекуррентное соотношение Беллмана*:

$$\varphi_1(y) = \begin{cases} c_1, & \text{если } a_1 \le y, \\ 0, & \text{иначе,} \end{cases} \qquad y = 0, \ldots, V. \quad (30)$$

$$\varphi_k(y) = \begin{cases} \max\{\varphi_{k-1}(y - a_k) + c_k,\ \varphi_{k-1}(y)\}, & \text{если } a_k \le y, \\ \varphi_{k-1}(y), & \text{иначе,} \end{cases} \qquad y = 0, \ldots, V, \quad k = 2, \ldots, n. \quad (31)$$

Построим алгоритм, в основе которого лежат соотношения (30)–(31). Как и в предыдущей задаче, значения функции $\varphi_k(y)$ будем записывать в специальном массиве φ. Основная часть алгоритма состоит из двух вложенных циклов. Во внешнем цикле переменная k пробегает значения от 1 до n, внутри него цикл по переменной y. При каждом k для определения значений $\varphi_k(y)$ необходимы значения этой функции, рассчитанные на предыдущей итерации внешнего цикла (значения $\varphi_{k-1}(y')$). При этом значения, полученные на более ранних итерациях внешнего цикла, не требуются. Более того, если внутренний цикл пустить по убыванию переменной y от V до 0, то получаемые значения функции $\varphi_k(y)$, без потери нужной информации, можно хранить в одномерном массиве $\varphi[0..V]$.

Запишем псевдокод алгоритма:

```
for y = 0 to V do
        φ[y]=0
for k = 1 to n do
        for y = V downto 0 do
                if (a_k ≤ y) and (φ[y] < φ[y − a_k] + c_k) then
                        φ(y) = φ[y − a_k] + c_k
                        ψ_k[y] = 1
                else
                        ψ_k[y]=0
```

Здесь $\psi_k[y]$ — вспомогательный двумерный массив (первый индекс k не помещен в квадратные скобки для наглядности, в псевдокоде такие вольности допускаются). Значения $\psi_k[y]$ определяются следующим образом. Если при $a_k \le y$ выполнено $\varphi_{k-1}(y - a_k) + c_k > \varphi_{k-1}(y)$ (т. е. $\varphi_k(y) = \varphi_{k-1}(y - a_k) + c_k$), то $\psi_k[y] = 1$. Иначе (при $\varphi_k(y) = \varphi_{k-1}(y)$) $\psi_k[y] = 0$.

По значениям массива ψ мы будем определять оптимальный набор предметов и соответствующие ему значения переменных. Для этого необходимо выполнить процедуру:

```
y = V
for k = n downto 1 do
        if ψ_k[y] = 1 then
                x_k = 1
                y = y − a_k
        else
                x_k = 0
```

Отметим, что временная сложность алгоритма решения задачи равна $O(V \cdot n)$. Не известно алгоритмов решения задачи о рюкзаке, которые имели бы полиномиальную сложность, зависящую только от n.

Рассмотрим пример. Пусть $V = 10$, $n = 4$, $a_1 = 6$, $a_2 = 5$, $a_3 = 4$, $a_4 = 2$, $c_1 = 3$, $c_2 = 4$, $c_3 = 5$, $c_4 = 1$.

y	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
$\varphi_1[y]$	0	0	0	0	0	0	3	3	3	3	3
$\psi_1[y]$	0	0	0	0	0	0	1	1	1	1	1
$\varphi_2[y]$	0	0	0	0	0	4	4	4	4	4	4
$\psi_2[y]$	0	0	0	0	0	1	1	1	1	1	1
$\varphi_3[y]$	0	0	0	0	5	5	5	5	5	9	9
$\psi_3[y]$	0	0	0	0	1	1	1	1	1	1	1
$\varphi_4[y]$	0	0	1	1	5	5	7	7	7	9	9
$\psi_4[y]$	0	0	1	1	0	0	1	1	1	0	0

Оптимальное значение целевой функции $\varphi_4[10] = 9$. Так как $\psi_4[10] = 0$, то четвертый предмет брать не нужно и $x_4 = 0$. Грузоподъемность рюкзака не изменилась и осталась равной 10. Значение $\psi_3[10] = 1$, значит, третий предмет помещаем в рюкзак ($x_3 = 1$), остаток грузоподъемности равен 6. По значению $\psi_2[6] = 1$ определяем $x_2 = 1$ — второй предмет добавляем к набору. Остаток грузоподъемности 1. Первый предмет не берем ввиду $\psi_1[1] = 0$ ($x_1 = 0$).

4.3 Выводы

Отметим свойства задач, для решения которых применимо динамическое программирование:

1. Свойство оптимальности для подзадач.

2. Наличие перекрывающихся подзадач.

Со свойством оптимальности для подзадач мы уже сталкивались при обсуждении жадных алгоритмов. Поэтому перейдем к обсуждению второго свойства. Для примера рассмотрим задачу линейного раскроя с двумя деталями с длинами $l_1 = 2$ и $l_2 = 3$. Пусть необходимо вычислить значения $\varphi[y]$ при $y = 8$ и $y = 7$. В соответствии с (28) имеем:

$$\varphi[8] = \max\{\varphi[6] + c_1,\ \varphi[5] + c_2\},$$

$$\varphi[7] = \max\{\varphi[5] + c_1,\ \varphi[4] + c_2\}.$$

Таким образом, значение $\varphi[5]$ используется как для вычисления $\varphi[8]$, так и для $\varphi[7]$. В этом случае можно сказать, что у двух подзадач есть одна общая подзадача. Если каждую встречающуюся подзадачу решать всякий раз заново, то общее число решенных подзадач станет огромным.

Разумнее решение каждой подзадачи записывать в таблицу и при необходимости в дальнейшем брать решения из нее.

Отметим, что при наличии перекрывающих подзадач крайне нежелательна реализация рекуррентных формул посредством рекурсивных функций. Напротив, лучше использовать циклические алгоритмы и массивы для хранения полученных результатов.

4.4 Задачи

1. Докажите рекуррентное соотношение (30)–(31).

2. Предположим, что необходимо перемножить k числовых матриц $M_1, M_2, \ldots, M_n$, где матрица M_i имеет размер m_i на n_i. Предположим, что для перемножения двух матриц размером p на q и q на r требуется $p \cdot q \cdot r$ операций. Найдите оптимальный порядок перемножения матриц, который минимизирует общее число операций.

3. Напишите алгоритм поиска для заданной числовой последовательности самой длинной неубывающей подпоследовательности.

4. Рассмотрим задачу о наибольшей общей подпоследовательности (НОП). Даны две последовательности $X = \langle x_1, \ldots, x_m \rangle$ и $Y = \langle y_1, \ldots, y_n \rangle$. Необходимо построить их общую подпоследовательность наибольшей длины.

   Назовем префиксом длины i последовательность $X_i = \langle x_1, \ldots, x_i \rangle$, $i = 0, 1, \ldots, m$. Пусть $c[i, j]$ — длина НОП для последовательностей X_i и Y_j. Для величин $c[i, j]$ можно вывести следующее рекуррентное соотношение:

   $$c[i,j] = \begin{cases} 0, & \text{если } i = 0 \text{ или } j = 0, \\ c[i-1, j-1] + 1, & \text{если } i, j > 0 \text{ и } x_i = y_j, \\ \max\{c[i, j-1], c[i-1, j]\}, & \text{если } i, j > 0 \text{ и } x_i \neq y_j. \end{cases} \tag{32}$$

   (a) Докажите рекуррентную формулу (32).

   (b) Напишите алгоритм, использующий формулу (32) и вычисляющий длину НОП X и Y.

   (c) Каким образом по заполненной таблице c (32) можно быстро построить НОП последовательностей X и Y?

5 Локальный поиск

Многие задачи решаются следующим способом. На первом шаге получают некоторое начальное решение, например, применением жадного алгоритма. Затем решение задачи ищут на множестве решений, "близких" к текущему. Полученное решение становится новым текущим решением. Этот шаг повторяют до тех пор, пока улучшается текущее решение.

Рассмотрим применение метода на примере решения задач о камнях и коммивояжера.

5.1 Задача о камнях

Пусть начальное решение получено при помощи некоторого алгоритма, например, применением жадной эвристики. Попробуем решение улучшить следующим образом. Будем менять местами камни из различных куч до тех пор, пока это приводит к улучшению решения. Приведем псевдокод алгоритма:

$$
\begin{array}{l}
rec = \max_{i=1,\ldots,m} \left\{ \sum_{j \in N_i} v_j \right\} \\
\text{for } (j_1, j_2) \in N \times N \\
\qquad i_1 = x_{j_1},\ i_2 = x_{j_2} \\
\qquad N_{i_1} = N_{i_1} \cup \{j_2\} \setminus \{j_1\} \\
\qquad N_{i_2} = N_{i_2} \cup \{j_1\} \setminus \{j_2\} \\
\qquad \text{if } rec \le \max_{i=1,\ldots,m} \left\{ \sum_{j \in N_i} v_j \right\} \text{ then} \\
\qquad\qquad rec = \max_{i=1,\ldots,m} \left\{ \sum_{j \in N_i} v_j \right\} \\
\qquad\qquad N_{i_1} = N_{i_1} \cup \{j_1\} \setminus \{j_2\} \\
\qquad\qquad N_{i_2} = N_{i_2} \cup \{j_2\} \setminus \{j_1\}
\end{array}
$$

Здесь N — множество камней, x_j — номер кучи, в которую помещен камень j. В этом алгоритме предполагается, что уже изначально получено некоторое начальное разбиение $N_1, \ldots, N_m$ камней на кучи. Затем перебираются все возможные пары камней. Для каждой пары камни меняются местами. Это изменение отвергается, если оно не приводит к уменьшению целевой функции.

5.2 Задача коммивояжера

Методы локального поиска часто применяют для решения задачи коммивояжера (в нашем случае с симметрической матрицей расстояний). Эту задачу можно интерпретировать как задачу поиска на взвешенном неориентированном графе гамильтонова цикла с наименьшей суммой весов ребер. Для поиска локольно-оптимального маршрута начинаем с какого-либо произвольного гамильтонова цикла и рассмотрим все пары содержащихся в нем несмежных ребер (A, B) и (C, D).

Локальное преобразование заключается в следующем. Удалим ребра (A, B) и (C, D). Начнем новый цикл с точки B, продолжим по части старого до точки C, включим ребро (C, A), затем по части старого цикла от A до D и добавим ребро (D, B).

Если цикл улучшается путем такого преобразования, то это нужно сделать, а затем продолжить рассмотрение оставшихся пар ребер.

6 Имитация отжига

Рассмотрим на примере задачи коммивояжера.

Введем две элементарные операции изменения текущего решения: перенос и обращение. При переносе часть маршрута вырезается и вставляется в другое место. При обращении выбирается фрагмент маршрута и порядок прохождения городов в нем меняется на противоположный. После применения одной из предложенных операций меняется значение целевой фунции на величину Δ. Это изменение применяется с вероятностью:

$$p = \begin{cases} 1, & \text{если } \Delta \leq 0, \\ e^{-\frac{\Delta}{T}}, & \text{если} \Delta > 0, \end{cases} \tag{33}$$

где T — значение температуры. Следовательно, возможен переход к решению с худшим значением целевой функции. Вероятность такого перехода тем больше, чем выше T и меньше Δ.

Работа алгоритма начинается с некоторого начального решения при высоком значении температуры T. Через каждые kN элементарных операций температура снижается на $p\%$. Поиск решения продолжается до тех пор, пока будут происходить изменения маршрута.

7 Ослабление задачи

Под ослаблением "трудной" задачи понимают снятие некоторых начальных ограничений, что делает задачу "легкой" для решения. При этом множество допустимых решений, как правило, возрастает.

Опишем формально. Дана задача P:

$$\varphi(x) \longrightarrow \min, \tag{34}$$

$$x \in \Omega. \tag{35}$$

Пусть Ω' — надмножество Ω ($\Omega \subset \Omega'$). Рассмотрим задачу P':

$$\varphi(x) \longrightarrow \min, \tag{36}$$

$$x \in \Omega'. \tag{37}$$

Очевидно, что если решение x^* задачи P' принадлежит множеству Ω ($x^* \in \Omega$), то x^* будет и решением задачи P.

Расмотрим на конкретном примере. Пусть дана задача линейного целочисленного программирования P:

$$\sum_{j=1}^{n} c_j x_j \longrightarrow \min, \tag{38}$$

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \qquad i = 1, \ldots, m, \tag{39}$$

$$x_j \geq 0, \quad \text{целые}, \quad j = 1, \ldots, n. \tag{40}$$

Если снять требование целочисленности переменных, то получится ослабленная задача — задача линейного программирования P':

$$\sum_{j=1}^{n} c_j x_j \longrightarrow \min, \tag{41}$$

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \qquad i = 1, \ldots, m, \tag{42}$$

$$x_j \geq 0, \qquad j = 1, \ldots, n. \tag{43}$$

На практике для ее решения эффективно применяется симплексный метод [3]. Если в результате решения задачи P' получено целочисленное решение, то оно же является и решением исходной задачи P.

Для некоторых частных случаев задач линейного программирования (например, транспортная задача с целочисленными параметрами), симплекс-метод всегда дает целочисленное решение. Для других задач, нецелые значения решения могут быть округлены. Но чаще всего ослабленную задачу P' используют для оценки решения задачи P, либо как элемент общей схемы решения.

8 Генетические алгоритмы

В основе алгоритма генетического типа лежит идея моделирования развития популяции живых организмов в процессе естественного отбора. Отдельные особи популяции по традиции будем называть хромосомами. Каждая хромосома соответствует одному возможному решению задачи.

Для реализации алгоритма необходимо описать операции кроссинговера (соответствует скрещиванию особей) и мутации (изменение свойств особи), определить фитнес-функцию (величина, характеризующая выживаемость особи), максимальный размер популяции и выработать стратегию управления популяцией.

8.1 Задача о рюкзаке

Пусть каждая хромосома s популяции кодируется последовательностью z^s из n элементов:

$$z^s = \{z_1^s, \ldots, z_j^n\}, \quad z_j^s \in \{0, 1\}, \quad j = 1, \ldots, n. \tag{44}$$

При этом должно выполнятся условие:

$$\sum_{j=1}^{n} a_j z_j^s \leq V. \tag{45}$$

Изначально необходимо сгенерировать каким-либо образом (случайно) p хромосом.

Фитнес-функция соответствует целевой функции задачи о рюкзаке:

$$\varphi(z^s) = \sum_{j=1}^{n} c_j z_j^s. \tag{46}$$

Определим операцию кроссинговер следующим образом:

Шаг 1. Выберем из популяции случайным образом t хромосом с номерами $s_1, \ldots, s_t$.

Шаг 2. Определим количество единиц на j-ом месте в выбранных решениях:

$$u_j = \sum_{k=1}^{t} z_j^{s_k}, \qquad j = 1, \ldots, n. \tag{47}$$

Шаг 3. Упорядочим величины u_j по невозрастанию значений:

$$u_{j_1} \leq \ldots \leq u_{j_n}. \tag{48}$$

Шаг 4. Будем назначать единицы в новом решении в соответствии с этим порядком:

$y = V$
for $q = 1$ to n do
    if $a_{j_q} \leq y$ then
        $z_{j_q} = 1$
        $y = y - a_{j_q}$
    else
        $z_{j_q} = 0$

Шаг 5. Удалим из популяции хромосому с худшим значением фитнес-функции.

Шаг 6. Добавим в популяцию полученное на шаге 4 решение $z = \{z_1, \ldots, z_j\}$.

Операция мутации соответствует локальным улучшениям решения z^s. Можно перебирать пары j_0, j_1 для которых $z_{j_0}^s = 0$, $z_{j_1}^s = 1$ и пытаться заменить единицу на ноль, а ноль на единицу: $z_{j_0}^s := 1$, $z_{j_1}^s := 0$. Такое замещение принимается, если приводит к увеличению значения фитнес-функции $\varphi(z^s)$.

В процессе работы генетического алгоритма последовательно применяются описанные операции кроссинговера и мутации случайно выбранного решения до тех пор, пока не выполнится один из критериев останова:

1) итекло допустимое время выполнения;

2) выполнено запланированное число итераций;

3) не происходит улучшения решения с максимальным значением фитнес-функции на после применения нескольких этих операций.

Могут быть и другие варианты реализации кроссинговера и мутации.

9 Заключение

В заключении подведем итоги и сделаем выводы.

Любая задача оптимизации с конечным множеством допустимых решений может быть решена методом полного перебора. Однако, на практике этот метод применим только для решения задач малой размерности. Для создания подобных алгоритмов требуется использование процедуры герерации некоторых комбинаторных объектов — перестановок, разбиений, подмножеств и т. п. При этом, для сокращения времени работы желательно, чтобы, во-первых, не рассматривались недопустимые с точки зрения ограничений задачи варианты решений, во-вторых, одни и те же допустимые решения не обрабатывалось дважды.

Часто при организации переборных алгоритмов решения задача разбивается на подзадачи, каждая из которых решается отдельно. Эти подзадачи разбиваются опять на подзадачи и так происходит далее, пока не будут получены подзадачи либо с малым временем решения, либо с единственным допустимым решением. Иногда для некоторых подзадач, используя специальные оценки, удается установить, что их решение не будет лучше некоторого текущего рекорда. Такие подзадачи не решаются и тем самым сокращается общее время работы алгоритма. Методы, основанные на подобной идее, называются методами ветвей и границ.

Если в процессе решения одни и те же подзадачи встречаются многократно, то имеет смысл решение каждой из них записывать в специальную таблицу и в следующий раз брать искомые значения из нее. Если при этом на основе оптимальных решений подзадач можно построить

оптимальное решение задачи, то в этом случае можно применять метод динамического программирования.

Для многих задач алгоритм решения устроен следующим образом. Начинают процесс с пустого решения. Затем на каждом шаге происходит локальное улучшение этого решения — в него вносятся "самые лучшие" имеющиеся фрагменты, которые впоследствии не удаляются. Так продолжается до тех пор, пока не будет построено допустимое решение задачи. Такой алгоритм называется жадным. Если жадный алгоритм приводит к оптимальному решению, то говорят что к задаче применим принцип жадного выбора. Такие алгоритмы, как правило, выполняются быстрее алгоритмов, основанных на динамическом программировании и гораздо быстрее чем полный перебор.

Для NP-полной задачи еще никому не удалось создать быстрый алгоритм решения. По-видимому, это невозможно, хотя это утверждение пока и не доказано. На практике такие задачи встречаются часто и их решать надо. Поэтому применяют так называемые эвристические методы. Как правило, они дают либо оптимальные, либо близкие к оптимальным решения и при этом быстро выполняются.

Решение, полученное эвристическим алгоритмом можно улучшить, применив метод локального поиска или имитации отжига.

Хорошо зарекомендовали себя генетические алгоритмы решения NP-полных задач.

Иногда часть ограничений NP-полной задачи можно снять получив при этом более простую (ослабленную) задачу. Если оптимальное решение последней удовлетворяет всем ограничениям исходной задачи, то оно является и ее оптимальным решением.

Выбор того или иного метода решения зависит от ряда условий. Для многих задач, ставших классическими, известны лучшие способы их решения. В списке рекомендуемой литературы приведены ссылки на некоторые источники, в которых содержатся описания многих полезных алгоритмов.

Заключение

В заключении подведем итоги и сделаем выводы.

Любая задача оптимизации с конечным множеством допустимых решений может быть решена методом полного перебора. Однако на прак-

тике этот метод применим только для решения задач малой размерности. Для создания подобных алгоритмов требуется использование процедуры генерации некоторых комбинаторных объектов — перестановок, разбиений, подмножеств и т. п. При этом для сокращения времени работы желательно, чтобы, во-первых, не рассматривались недопустимые с точки зрения ограничений задачи варианты решений, во-вторых, одни и те же допустимые решения не обрабатывались дважды.

Часто при организации переборных алгоритмов решения задача разбивается на подзадачи, каждая из которых решается отдельно. Эти подзадачи разбиваются опять на подзадачи и так происходит далее, пока не будут получены подзадачи либо с малым временем решения, либо с единственным допустимым решением. Иногда для некоторых подзадач, используя специальные оценки, удается установить, что их решение не будет лучше некоторого текущего рекорда. Такие подзадачи не решаются, и тем самым сокращается общее время работы алгоритма. Методы, основанные на подобной идее, называются методами ветвей и границ.

Если в процессе решения одни и те же подзадачи встречаются многократно, то имеет смысл решение каждой из них записывать в специальную таблицу и в следующий раз брать искомые значения из нее. Если при этом на основе оптимальных решений подзадач можно построить оптимальное решение задачи, то в этом случае можно применять метод динамического программирования.

Для многих задач алгоритм решения устроен следующим образом. Начинают процесс с пустого решения. Затем на каждом шаге происходит локальное улучшение этого решения — в него вносятся “самые лучшие” имеющиеся фрагменты, которые впоследствии не удаляются. Так продолжается до тех пор, пока не будет построено допустимое решение задачи. Такой алгоритм называется жадным. Если жадный алгоритм приводит к оптимальному решению, то говорят, что к задаче применим принцип жадного выбора. Такие алгоритмы, как правило, выполняются быстрее алгоритмов, основанных на динамическом программировании, и гораздо быстрее, чем полный перебор.

Для NP-полной задачи еще никому не удалось создать быстрый алгоритм решения. По-видимому, это невозможно, хотя это утверждение пока и не доказано. На практике такие задачи встречаются часто и их решать надо. Поэтому применяют так называемые эвристические методы. Как правило, они дают либо оптимальные, либо близкие к оптимальным решения и при этом быстро выполняются.

Решение, полученное эвристическим алгоритмом, можно улучшить, применив метод локального поиска.

Иногда часть ограничений NP-полной задачи можно снять, получив при этом более простую (ослабленную) задачу. Если оптимальное решение последней удовлетворяет всем ограничениям исходной задачи, то оно является и ее оптимальным решением.

Выбор того или иного метода решения зависит от ряда условий. Для многих задач, ставших классическими, известны лучшие способы их решения. В списке рекомендуемой литературы приведены ссылки на некоторые источники, в которых содержатся описания многих полезных алгоритмов.

Список литературы

[1] Ахо А. В. Построение и анализ вычислительных алгоритмов / А. В. Ахо, Д. Э. Хопкрофт, Д. Д. Ульман. М.: Мир, 1979.

[2] Ахо А. В. Структуры данных и алгоритмы / А. В. Ахо, Д. Э. Хопкрофт, Д. Д. Ульман. М.: Издательский дом "Вильямс", 2003.

[3] Данциг Дж. Б. Линейное программирование, его применения и обобщения / Дж. Б. Данциг. М.: Прогресс, 1966.

[4] Ковалев М. М. Дискретная оптимизация (целочисленное программирование) / М. М. Ковалев. М.: Едиториал УРСС, 2003.

[5] Кормен Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. М.: МЦНМО: БИНОМ. Лаборатория знаний, 2004.

[6] Липский В. Комбинаторика для программистов / В. Липский. М.: Мир, 1988.

[7] Новиков Ф. А. Дискретная математика для программистов / Ф. А. Новиков. СПб.: Питер, 2003.

[8] Романовский И. В. Дискретный анализ / И. В. Романовский. СПб.: Невский диалект, 2003.

[9] Романовский И. В. Субоптимальные решения / И. В. Романовский. Петрозаводск: Изд-во ПетрГУ, 1998.

[10] Сачков В. Н. Введение в комбинаторные методы дискретной математики / В. Н. Сачков. М.: Наука, 1982.

[11] Сигал И. Х. Введение в прикладное дискретное программирование: модели и вычислительные алгоритмы / И. Х. Сигал, А. П. Иванова. М.: ФИЗМАТЛИТ, 2002.

[12] Сухарев А. Г. Курс методов оптимизации / А. Г. Сухарев, А. В. Тимохов, В. В. Федоров. М.: Изд-во Наука, 1986.