

PACS 07.05.Kf, 02.10.Ox, 07.05.Fb

© 2007 г. А.А. ЛАЗАРЕВ, канд. физ.-мат. наук
(Вычислительный центр РАН, Москва)

ГРАФИЧЕСКИЙ ПОДХОД К РЕШЕНИЮ ЗАДАЧ КОМБИНАТОРНОЙ ОПТИМИЗАЦИИ¹

Рассматривается графическая реализация метода динамического программирования. Идея метода показана на примерах решения задач **разбиения** и **рюкзак**. Проведен сравнительный анализ предлагаемого метода с известными алгоритмами решения этих задач.

1. Введение

В известных работах [1, 2] рассматриваются классические NP – трудные в обычном смысле задачи комбинаторной оптимизации: **разбиения** и **рюкзак**. Здесь будем рассматривать следующие их формулировки.

Задача разбиения (partition). Задано упорядоченное множество из n положительных целых чисел $B = \{b_1, \dots, b_n\}$, $b_1 \geq \dots \geq b_n$. Требуется разбить множество B на два подмножества B_1 и B_2 , так чтобы минимизировать значение:

$$(1) \quad \left| \sum_{b_i \in B_1} b_i - \sum_{b_i \in B_2} b_i \right| \rightarrow \min.$$

Задача об одномерном рюкзаке (knapsack). Задачу можно представить в виде задачи целочисленного линейного программирования:

$$(2) \quad \begin{cases} f(x) = \sum_{i=1}^n c_i x_i \rightarrow \max \\ \sum_{i=1}^n a_i x_i \leq A, \\ x_i \in \{0, 1\}, i = 1, \dots, n. \end{cases}$$

Когда $c_i = a_i = b_i$, $i = 1, \dots, n$, и $A = \frac{1}{2} \sum_{j=1}^n b_j$, то задачи (1) и (2) являются эквивалентными.

¹ Работа выполнена в рамках научной школы академика Ю.И. Журавлёва, НШ-5833.2006.1.

2. Графический алгоритм для задачи разбиения

В алгоритме последовательно на шаге $\alpha = 1, \dots, n$ рассматриваются следующие функции:

$$F_\alpha^1(t) = \left| \sum_{b_j \in B_1(t-b_\alpha)} b_j - \sum_{b_j \in B_2(t-b_\alpha)} b_j \right|;$$

$$F_\alpha^2(t) = \left| \sum_{b_j \in B_1(t+b_\alpha)} b_j - \sum_{b_j \in B_2(t+b_\alpha)} b_j \right|.$$

В функции $B_1(t)$ (аналогично в $B_2(t)$) ставится в соответствие каждой точке t из рассматриваемого интервала $\left[-\sum_{j=1}^{\alpha} b_j, \sum_{j=1}^{\alpha} b_j \right]$ некоторое множество элементов $\overline{B_1}$ (соответственно, $\overline{B_2}$). Стоит отметить, что на каждом шаге α алгоритма $B_1(t) \cup B_2(t) = \{b_1, \dots, b_{\alpha-1}\} = \overline{B_1} \cup \overline{B_2}, \forall t$.

2.1. Идея графического алгоритма

Последовательно выбирается “включение” очередного числа b_α , где $\alpha = 1, \dots, n$, в множество $B_1(t)$ или $B_2(t)$. В каждой точке t “включение” числа b_α выбирается так, чтобы минимизировать значение функции $\left| \sum_{b_j \in B_1(t)} b_j + t - \sum_{b_j \in B_2(t)} b_j \right|$, где $t = t_1 - t_2$, t_1 – величина, которая будет добавлена к $B_1(t)$, t_2 – к $B_2(t)$. Параметр t характеризует сумму чисел, которая будет добавлена к $B_1(t)$ на последующих шагах. Если $t < 0$, то $-t$ равно сумме чисел, которая будет добавлена к $B_2(t)$ на последующих шагах. На очередном шаге α из функции $F_{\alpha-1}(t) = \left| \sum_{b_j \in B_1(t)} b_j - \sum_{b_j \in B_2(t)} b_j \right|$, полученной на предыдущем $\alpha - 1$ шаге, строится функция

$$F_\alpha(t) := \min\{F_{\alpha-1}(t - b_\alpha), F_{\alpha-1}(t + b_\alpha)\} = \min\{F_\alpha^1(t), F_\alpha^2(t)\},$$

где $F_0(t) := 0, \forall t$. Если $F_\alpha^1(t) < F_\alpha^2(t)$, то $B_1(t) := B_1(t - b_\alpha) \cup \{b_\alpha\}$, иначе $B_2(t) := B_2(t + b_\alpha) \cup \{b_\alpha\}$. Кусочно-линейную функцию $F_\alpha(t)$ можно представить (и хранить) в табличном виде по точкам “излома”: $t_0; t_1; \dots; t_{m_\alpha}$. На промежутке $\left[t_i - \frac{t_i - t_{i-1}}{2}, t_i + \frac{t_{i+1} - t_i}{2} \right)$, $i = 1, \dots, m_\alpha - 1$, кусочно-линейная функция $F_\alpha(t)$ задается уравнением $F_\alpha(t) = |t - t_i|$, т.е. график функции пересекает ось t в точке t_i . Каждому временному интервалу $\left[t_i - \frac{t_i - t_{i-1}}{2}, t_i + \frac{t_{i+1} - t_i}{2} \right)$ соответствует некоторое фиксированное разбиение $(\overline{B_1}; \overline{B_2})$, т.е. $B_1(t^1) = B_1(t^2) = \overline{B_1}, \forall t^1, t^2 \in \left[t_i - \frac{t_i - t_{i-1}}{2}, t_i + \frac{t_{i+1} - t_i}{2} \right)$ (аналогично и для $B_2(t)$). Пусть на предыдущем шаге $\alpha - 1$ получена функция $F_{\alpha-1}(t)$, заданная таблично. На шаге α рассматриваем две функции $F_{\alpha-1}(t - b_\alpha)$ и $F_{\alpha-1}(t + b_\alpha)$, задающиеся соответственно двумя таблицами в точках “излома”: $t_0 - b_\alpha, \dots, t_i - b_\alpha, \dots, t_{m_{\alpha-1}} - b_\alpha$ и $t_0 + b_\alpha, \dots, t_i + b_\alpha, \dots, t_{m_{\alpha-1}} + b_\alpha$. Сопоставляя графики обеих функций, рассматриваем временные промежутки $[t^1, t^2], t^1, t^2 \in \{t_0 - b_\alpha, \dots, t_{m_{\alpha-1}} - b_\alpha, t_0 + b_\alpha, \dots, t_{m_{\alpha-1}} + b_\alpha\}$, на которых функция

$F_{\alpha-1}(t - b_\alpha)$ (и, соответственно, $F_{\alpha-1}(t + b_\alpha)$) задается одним уравнением кусочно-линейной функции. На данном промежутке кусочно-линейные функции $|t - a|$ и $|t - b|$ пересекаются не более чем в одной точке (или совпадают).

Очевидно, что функция $F_\alpha(t)$ может быть задана таблицей, состоящей из множества точек $\{t_0 - b_\alpha, \dots, t_{m_{\alpha-1}} - b_\alpha, t_0 + b_\alpha, \dots, t_{m_{\alpha-1}} + b_\alpha\}$, которое будет упорядочено по неубыванию, часть точек может сократиться. Рассматриваются интервалы $[t^1, t^2] \in \left[-\sum_{j=1}^n b_j, \sum_{j=1}^n b_j \right]$. Следовательно, функция $F_\alpha(t)$ будет задаваться не более чем $m_\alpha = 2 * m_{\alpha-1}$ точками.

Решением задачи является разбиение $(B_1(0), B_2(0))$, полученное на последнем шаге $\alpha = n$. Значение целевой функции равно $F_n(0)$.

2.2. Сокращение рассматриваемых интервалов

Так как на шаге n требуется вычислить значение целевой функции и найти разбиение лишь в точке $t = 0$, то на шаге $n - 1$ нам достаточно рассчитать значения в точках $t \in [-b_n, b_n]$. Аналогично на шаге $n - 2$ достаточно рассмотреть интервал $[-b_n - b_{n-1}, b_n + b_{n-1}]$ и т.д.

Следовательно, на каждом шаге α достаточно рассматривать интервал $\left[-\sum_{j=\alpha+1}^n b_j, \sum_{j=\alpha+1}^n b_j \right]$ вместо интервала $\left[-\sum_{j=1}^n b_j, \sum_{j=1}^n b_j \right]$.

При построении функции $F_\alpha(t)$ рассматриваются только точки $t \in \left[-\sum_{j=\alpha+1}^n b_j, \sum_{j=\alpha+1}^n b_j \right]$. Чтобы интервал сокращался максимально быстро, необходимо упорядочить b_j по невозрастанию. Если учесть, что функция $F_\alpha(t)$, $\alpha = 1, \dots, n$, является четной, то при практической реализации алгоритма достаточно хранить "половину" таблицы.

2.3. Пример

Рассмотрим пример $B = \{100, 70, 50, 20\}$, $n = 4$. Числа пронумерованы по невозрастанию.

Шаг 0. $F_0(t) := 0$, $B_1(t) = \emptyset$, $B_2(t) = \emptyset$, $\forall t$.

Шаг 1. Размещение для $b_1 = 100$. Рассматриваются 2 точки: $0 + 100$ и $0 - 100$. Сравнение функций происходит на трех интервалах $[-240, -100]$, $[-100, 100]$, $[100, 240]$ (или, за счет сокращения интервалов, на $[-140, -100]$, $[-100, 100]$, $[100, 140]$), где $240 = \sum_{j=1}^n b_j$.

На интервале $[-240, 0]$ имеем оптимальное разбиение $B_1(t) = \{b_1\}$, $B_2(t) = \emptyset$, на интервале $[0, 240]$ оптимальное разбиение $B_1(t) = \emptyset$, $B_2(t) = \{b_1\}$. Результаты вычислений и целевая функция $F_1(t)$ (в расширенном виде) представлены на рис. 1. Храним следующую информацию:

-100	100
(100;)	(; 100)

Как отмечалось выше, достаточно хранить только "половину" таблицы.

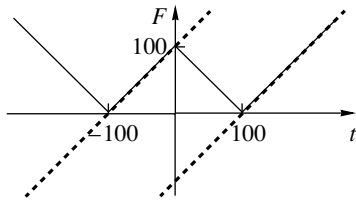


Рис. 1. Функция $F_1(t)$.

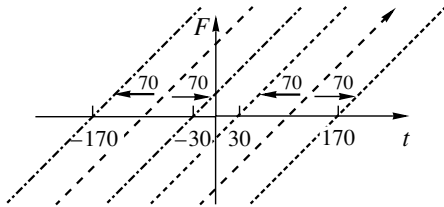


Рис. 2. Преобразование функции $F_1(t)$.

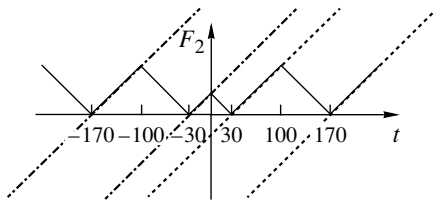


Рис. 3. Функция $F_2(t)$.

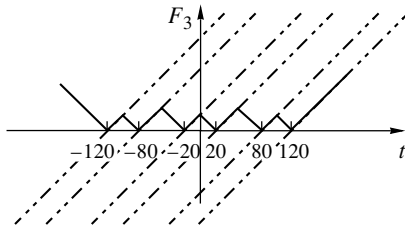


Рис. 4. Функция $F_3(t)$.

Шаг 2. Размещение для $b_2 = 70$. Рассматриваются 4 точки: $-100 - 70 = -170$; $-100 + 70 = -30$; $100 - 70 = 30$; $100 + 70 = 170$. Вычисления производятся на пяти интервалах $[-240, -170]$, $[-170, -30]$, $[-30, 30]$, $[30, 170]$, $[170, 240]$. Но за счет сокращения достаточно рассмотреть только три интервала: $[-70, -30]$, $[-30, 30]$, $[30, 70]$. На интервале $[-70, 0]$ имеем оптимальное разбиение $B_1(t) = \{b_1\}$, $B_2(t) = \{b_2\}$, на интервале $[0, 70]$ оптимальное разбиение $B_1(t) = \{b_2\}$, $B_2(t) = \{b_1\}$. Фактически не рассматриваем интервалы, а сразу конструируем функцию $F_2(t)$, т.е. включаем в таблицу точки -30 и 30 и соответствующие им разбиения. Точке -30 соответствует разбиение $B_1(t) = \{b_1\}$, $B_2(t) = \{b_2\}$, а точке 30 – разбиение $B_1(t) = \{b_2\}$, $B_2(t) = \{b_1\}$. Преобразование функции $F_1(t)$ к функциям $F^1(t)$ и $F^2(t)$ представлено на рис. 2. Результаты вычислений и целевая функция $F_2(t)$ (в расширенном виде) представлены на рис. 3. Таким образом, для выполнения следующего шага алгоритма достаточно сохранять информацию только в одной точке $t = 30$:

30
(70; 100)

Шаг 3. Размещение для числа $b_3 = 50$. Рассматриваются четыре точки: $-30 - 50 = -80$; $-30 + 50 = 20$; $30 + 50 = 80$; $30 - 50 = -20$. За счет сокращения достаточно рассмотреть один интервал $[-20, 20]$. Результаты вычислений и целевая функция $F_3(t)$ (в расширенном виде) представлены на рис. 4. Храним следующую информацию для одной точки $t = 20$:

20
(70, 50; 100)

На шаге 4 получаем два оптимальных “симметричных” решения $B_1(0) = \{b_1, b_4\}$,

$B_2(0) = \{b_2, b_3\}$ и $B_1(0) = \{b_2, b_3\}$, $B_2(0) = \{b_1, b_4\}$.

Таким образом, рассмотрено точек: $2(-100 \& 100) + 2(-30 \& 30) + 2(-20 \& 20) + 1(0) = 7$, в то время как алгоритмом динамического программирования [3] (с сокращением интервалов) пришлось бы просмотреть $280 + 140 + 40 + 0 = 460$ точек. Алгоритм *Balsub* [2, стр. 83] с трудоемкостью $O(nb_{\max})$ операций, который является наилучшим из точных алгоритмов решения задачи разбиения, найдет решение за $2 \cdot \frac{n}{2} \cdot b_{\max} = 400$ операций.

Так как функция $F_\alpha(t)$ четная, то достаточно хранить лишь “половину” таблицы. Кроме того, следует отметить, что при “масштабировании с небольшим изменением

параметров” примера, т.е. когда $b'_j = Kb_j + \varepsilon_j$, где $|\varepsilon_j| \ll K$, K – некоторая достаточно большая положительная константа, $j = 1, \dots, n$, трудоемкость алгоритма, построенного на методе динамического программирования, составит $O(Kn \sum b_j)$ операций, в то время как у графического алгоритма трудоемкость не изменится. Для алгоритма *Balsub* с трудоемкостью $O(nb_{\max})$ операций такое “масштабирование” примера также приведет к увеличению трудоемкости в K раз. Таким образом, графический алгоритм находит решение за одно и то же количество операций для всех точек некоторого конуса в n -мерном пространстве, если представить параметры примера как точку (b_1, \dots, b_n) в n -мерном пространстве. Причем параметры примера могут быть как отрицательными, так и нецелочисленными.

2.4. Трудоемкость алгоритма

Теорема 1. Графический алгоритм находит на шаге $\alpha = n$ оптимальное разбиение $B_1(0)$ и $B_2(0)$.

Доказательство. Покажем, что на каждом шаге $\alpha = 1, \dots, n$ в каждой точке $t \in \left[-\sum_{j=\alpha+1}^n b_j, \sum_{j=\alpha+1}^n b_j \right]$ алгоритмом находится оптимальное частичное разбиение $B_1(t)$ и $B_2(t)$ подмножества чисел $\{b_1, \dots, b_\alpha\}$.

Доказательство проведем по методу математической индукции.

1. Очевидно, что на шаге $\alpha = 1$ в каждой точке $t \in \left[-\sum_{j=2}^n b_j, \sum_{j=2}^n b_j \right]$ получаем оптимальное разбиение $B_1(t)$ и $B_2(t)$.

2. Предположим, что на шаге α в каждой точке $t \in \left[-\sum_{j=\alpha+1}^n b_j, \sum_{j=\alpha+1}^n b_j \right]$ имеем некоторое оптимальное разбиение $B_1(t)$ и $B_2(t)$.

3. Покажем, что на шаге $\alpha + 1$ в каждой точке $t \in \left[-\sum_{j=\alpha+2}^n b_j, \sum_{j=\alpha+2}^n b_j \right]$ алгоритмом будет получено оптимальное разбиение $B_1(t)$ и $B_2(t)$.

От противного. Пусть в точке t алгоритмом рассмотрено два разбиения $(B_1(t - b_{\alpha+1}); B_2(t - b_{\alpha+1}) \cup \{b_{\alpha+1}\})$ и $(B_1(t + b_{\alpha+1}) \cup \{b_{\alpha+1}\}; B_2(t + b_{\alpha+1}))$.

В алгоритме выбирается то разбиение, на котором достигается минимальное значение величины $\left| \sum_{b_j \in B_1} b_j + t - \sum_{b_j \in B_2} b_j \right|$.

Пусть в точке t существует такое разбиение $(\bar{B}_1; \bar{B}_2)$, что выполняется

$$\left| \sum_{b_j \in B_1(t+b_{\alpha+1}) \cup \{b_{\alpha+1}\}} b_j + t - \sum_{b_j \in B_2(t+b_{\alpha+1})} b_j \right| > \left| \sum_{b_j \in \bar{B}_1} b_j + t - \sum_{b_j \in \bar{B}_2} b_j \right|$$

и

$$\left| \sum_{b_j \in B_1(t-b_{\alpha+1})} b_j + t - \sum_{b_j \in B_2(t-b_{\alpha+1}) \cup \{b_{\alpha+1}\}} b_j \right| > \left| \sum_{b_j \in \bar{B}_1} b_j + t - \sum_{b_j \in \bar{B}_2} b_j \right|.$$

Пусть $b_{\alpha+1} \in \overline{B}_1$, тогда имеем:

$$\left| \sum_{b_j \in B_1(t+b_\alpha)} b_j + b_{\alpha+1} + t - \sum_{b_j \in B_2(t+b_\alpha)} b_j \right| > \left| \sum_{b_j \in \overline{B}_1 \setminus \{b_{\alpha+1}\}} b_j + b_{\alpha+1} + t - \sum_{b_j \in \overline{B}_2} b_j \right|,$$

но тогда полученное на шаге α в точке $t + b_\alpha$ разбиение $(B_1(t + b_\alpha); B_2(t + b_\alpha))$ не оптимальное, так как разбиение $(\overline{B}_1 \setminus \{b_{\alpha+1}\}; \overline{B}_2)$ “лучше”. Получили противоречие. Аналогично может быть показано и для случая $b_{\alpha+1} \in \overline{B}_2$. \square

Из анализа графического алгоритма можно сделать следующие выводы:

1. Существует класс целочисленных примеров, на котором количество рассматриваемых точек от шага к шагу растет экспоненциальным образом. Например, $B = \{b_1, \dots, b_n\} = \{M, M-1, M-2, \dots, 1, 1, \dots, 1\}$. M – достаточно большое число, сумма единиц в примере равна $M(M+1)/2$, т.е. $n = M + M(M+1)/2$.

2. Существует класс нецелочисленных примеров $B = \{b_1, \dots, b_n\}$, на котором количество рассматриваемых точек от шага к шагу растет экспоненциальным образом. Например, если не существует такого набора чисел $\lambda_i = \pm 1$, $i = 1, \dots, n$, что выполняется $\lambda_1 b_1 + \dots + \lambda_n b_n = 0$, то количество точек в таком примере растет $O(2^n)$.

2.5. Экспериментальная оценка трудоемкости алгоритма

Был проведен сравнительный анализ алгоритмов, представленных в известной работе [2], с предлагаемым графическим алгоритмом. Проведены две группы экспериментов:

1. В первой группе экспериментов для всех целочисленных значений, когда параметры удовлетворяют ограничениям: $40 \geq b_1 \geq \dots \geq b_n \geq 1$, при $n = 4, 5, \dots, 10$. Полученные результаты сведены в табл. 1:

В первом столбце – размерность задачи (n); во втором – общее количество решенных примеров для данного n (число сочетаний из $b_{\max} + n - 1$ по n , где $b_{\max} = 40$); в третьем – среднее значение трудоемкости графического алгоритма; в четвертом – среднее значение трудоемкости алгоритма Valsub; в пятом – среднее значение трудоемкости алгоритма, основанного на методе динамического программирования; в шестом – максимальное значение трудоемкости графического алгоритма; в седьмом – максимальное значение трудоемкости алгоритма Valsub; в восьмом – максимальное значение трудоемкости алгоритма, основанного на методе динамического программирования; в девятом – количество примеров, для которых трудоемкость алгоритма Valsub меньше графического алгоритма; в десятом – количество примеров, для которых трудоемкость алгоритма, основанного на методе динамического программирования, меньше алгоритма Valsub.

2. Во второй группе экспериментов для $n = 4, 5, \dots, 10$ строились по 20 000 примеров, когда параметры примера выбирались равномерно $b_i \in [1, 200]$, $i = 1, \dots, n$. За-

Таблица 1

1	2	3	4	5	6	7	8	9	10
4	123 410	9	307	328	20	443	640	2	63 684
5	1 086 008	16	444	512	40	564	1000	2	337 077
6	8 145 060	29	542	738	60	687	1440	4	1 140 166
7	53 524 680	48	633	1004	140	811	1960	11	2 799 418
8	314 457 495	76	725	1312	212	933	2560	23	5 348 746
9	1 677 106 640	115	814	1660	376	1053	3240	83	8 488 253
10	8 217 822 536	168	905	2050	500	1172	4000	416	11 426 171

Таблица 2

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
4	8	1463	1591	16	2196	3080	4	2	6	1310	1604	20	2207	3200	10504	8970
5	16	2191	2490	40	2797	44675	8	3	17	2482	3102	40	2811	5000	6641	3642
6	29	2700	3586	60	3401	6570	12	4	26	2884	3932	60	3418	7176	3000	3170
7	50	3145	4881	140	4006	8729	15	6	54	3353	6794	136	4029	9800	1101	88
8	87	3600	6362	216	4617	11056	19	8	82	3849	7039	220	4645	12112	333	377
9	149	4050	8059	464	5241	13644	52	21	144	4109	11803	476	5232	16200	86	1
10	245	4499	9930	656	5815	16730	24	10	240	4732	12410	676	5854	18840	18	3

тем для каждого примера в n -мерном пространстве в окрестности $r = 100 + n$ решались $1000n$ примеров $\{b'_1, \dots, b'_n\}$ таким образом, что $b_i - (100 + n) \leq b'_i \leq b_i + (100 + n)$, $i = 1, \dots, n$. Причем если в окрестности находился пример с большей трудоемкостью, то “переходим в этот пример”. Процесс останавливался, когда в окрестности не удается найти “более сложные примеры”. Были получены следующие результаты (см. табл. 2).

В первом столбце – размерность задачи (n); во втором – четвертом столбцах – среднее значение трудоемкости алгоритмов: графического, Balsub и алгоритма, основанного на методе динамического программирования в “начальной точке”; в пятом – седьмом столбцах – максимальное значение алгоритмов в “начальной точке”; в 8–9-м – максимальное и среднее количество “переходов от начальной к конечной точке”; в 10–12-м – среднее значение трудоемкости исследуемых алгоритмов в “конечной точке”; в 13–15-м – максимальное значение трудоемкости исследуемых алгоритмов в “конечной точке”; в 16–17-м – количество примеров, для которых трудоемкость алгоритма, основанного на методе динамического программирования, меньше трудоемкости алгоритма Balsub, соответственно в начальных и конечных точках.

Во всех (!) примерах (“начальных” и “конечных”) трудоемкость алгоритма Balsub была больше, чем у графического алгоритма, за исключением 38 примеров для $n = 4$ и двух примеров для $n = 5$ из 20000 “конечных” примеров.

3. Графический подход для задачи одномерного рюкзака (0 – 1 knapsack)

3.1. Алгоритм динамического программирования для задачи рюкзака

Наиболее эффективным для задачи считается алгоритм динамического программирования, основанный на принципе оптимальности Беллмана [1, 3]. Алгоритм работает только в случае, когда параметры $A, a_i \in Z^+$, $i = 1, \dots, n$. На каждом шаге $\alpha = 1, \dots, n$ строится функция

$$g_\alpha(t) = \max_{x_\alpha \in \{0,1\}} (c_\alpha x_\alpha + g_{\alpha-1}(t - a_\alpha x_\alpha)), \quad t \geq a_\alpha x_\alpha,$$

в каждой точке $0 \leq t \leq A$. Для каждой точки t фиксируется соответствующий $x_\alpha = \arg \max g_\alpha(t)$. На шаге $\alpha = n$ в точке $t = A$ находим оптимальное решение.

Продемонстрируем работу алгоритма на примере из [4, стр. 125–129].

$$(3) \quad \begin{cases} f(x) = 5x_1 + 7x_2 + 6x_3 + 3x_4 \longrightarrow \max \\ 2x_1 + 3x_2 + 5x_3 + 7x_4 \leq 9, \\ x_i \in \{0, 1\}, \quad i = 1, \dots, 4. \end{cases}$$

Работу алгоритма можно представить в виде табл. 3.

Таблица 3

t	$g_1(t)$	$x(t)$	$g_2(t)$	$x(t)$	$g_3(t)$	$x(t)$	$g_4(t)$	$x(t)$
0	0	(0,,)	0	(0,0,,)	0	(0,0,0,)	0	(0,0,0,0)
1	0	(0,,)	0	(0,0,,)	0	(0,0,0,)	0	(0,0,0,0)
2	5	(1,,)	5	(1,0,,)	5	(1,0,0,)	5	(1,0,0,0)
3	5	(1,,)	7	(0,1,,)	7	(0,1,0,)	7	(0,1,0,0)
4	5	(1,,)	7	(0,1,,)	7	(0,1,0,)	7	(0,1,0,0)
5	5	(1,,)	12	(1,1,,)	12	(1,1,0,)	12	(1,1,0,0)
6	5	(1,,)	12	(1,1,,)	12	(1,1,0,)	12	(1,1,0,0)
7	5	(1,,)	12	(1,1,,)	12	(1,1,0,)	12	(1,1,0,0)
8	5	(1,,)	12	(1,1,,)	13	(0,1,1,)	13	(0,1,1,0)
9	5	(1,,)	12	(1,1,,)	13	(0,1,1,)	13	(0,1,1,0)

Таким образом, получаем оптимальное решение $(0, 1, 1, 0)$ и соответствующее значение целевой функции $g_4(9) = 13$. Трудоемкость алгоритма равна $O(nA)$ операций.

3.2. Графический подход

Функцию $g_\alpha(t)$ можно представить в следующем виде:

t	t_0	t_1	\dots	t_{m_α}
g	f_0	f_1	\dots	f_{m_α}

т.е. при $t \in [t_j, t_{j+1})$ имеем $g_\alpha(t) = f_j, j = 0, 1, \dots, m_\alpha - 1$.

Функцию $g_{\alpha+1}(t)$ можно получить из функции $g_\alpha(t)$ следующим образом:

$$\begin{aligned}
 g^1(t) &= g_\alpha(t), \quad x_{\alpha+1}(t) = 0, \\
 g^2(t) &= c_{\alpha+1} + g_\alpha(t - a_{\alpha+1}), \quad x_{\alpha+1}(t) = 1, \quad a_{\alpha+1} \leq t, \\
 g^2(t) &= g^1(t), \quad x_{\alpha+1}(t) = 0, \quad a_{\alpha+1} > t, \\
 g_{\alpha+1}(t) &= \max\{g^1(t), g^2(t)\}.
 \end{aligned}$$

График $g^2(t)$ строится из графика $g_\alpha(t)$ смещением “наверх” на $c_{\alpha+1}$ и “вправо” на $a_{\alpha+1}$. Функцию $g^2(t)$ можно представить в следующем виде

t	$t_0 + a_{\alpha+1}$	$t_1 + a_{\alpha+1}$	\dots	$t_{m_\alpha} + a_{\alpha+1}$
g	$f_0 + c_{\alpha+1}$	$f_1 + c_{\alpha+1}$	\dots	$f_{m_\alpha} + c_{\alpha+1}$

График $g^1(t)$ полностью повторяет график $g_\alpha(t)$. Следовательно, чтобы построить $g_{\alpha+1}(t) = \max\{g^1(t), g^2(t)\}$, необходимо рассмотреть не более $2m_\alpha$ интервалов, образованных точками из множества $\{t_0, t_1, \dots, t_{m_\alpha}, t_0 + a_{\alpha+1}, t_1 + a_{\alpha+1}, \dots, t_{m_\alpha} + a_{\alpha+1}\}$, которые принадлежат интервалу $[0, A]$. Количество точек не превосходит A для $a_i \in Z^+, i = 1, \dots, n$. Таким образом, трудоемкость графического алгоритма для целочисленного примера не превосходит $\min\{O(nA), O(nf_{\max})\}$, как и для алгоритма, основанного на методе динамического программирования. Необходимо также отметить, что параметры задачи могут быть как нецелочисленными, так и отрицательными. В этом случае график $g^2(t)$ строится из графика $g_\alpha(t)$ смещением “вниз” на $|c_{\alpha+1}|$ (когда $c_{\alpha+1} < 0$) и “влево” на $|a_{\alpha+1}|$ ($a_{\alpha+1} < 0$).

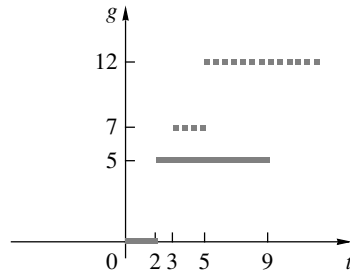


Рис. 5. Функции $g^1(t)$ и $g^2(t)$ (пунктиром).

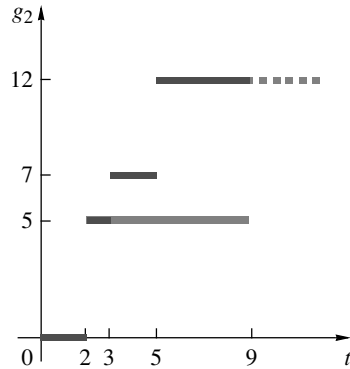


Рис. 6. Целевая функция $g_2(t)$.

Покажем на том же примере (3) работу графического алгоритма.

Шаг 1. В результате получаем следующие значения:

t	0	2
g	0	5
$x(t)$	(0, , ,)	(1, , ,)

Шаг 2. На рис. 5 представлены функции $g^1(t)$ и $g^2(t)$ (пунктиром). Для построения функции $g_2(t)$ необходимо рассмотреть интервалы, образованные точками 0, 2, 0 + 3, 2 + 3.

Результаты вычислений и целевая функция $g_2(t)$ представлены на рис. 6. В результате получим:

t	0	2	3	5
g	0	5	7	12
$x(t)$	(0, 0, ,)	(1, 0, ,)	(0, 1, ,)	(1, 1, ,)

Шаг 3. Для построения функции $g_3(t)$ необходимо рассмотреть интервалы, образованные точками 0, 2, 3, 5, 0 + 5, 2 + 5, 3 + 5. Точка 5 + 5 > 9 не рассматривается. Многие фрагменты $g^2(t)$ (обозначено пунктиром) “поглощаются” и не участвуют

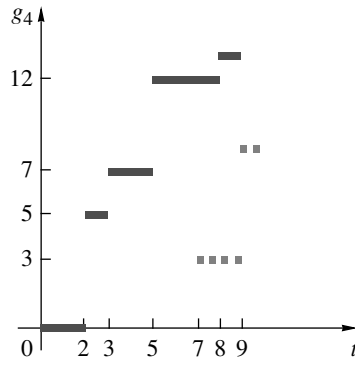


Рис. 7. Целевая функция $g_4(t)$.

в $g_3(t)$. Результатом работы третьего шага алгоритма будет:

t	0	2	3	5	8
g	0	5	7	12	13
$x(t)$	(0, 0, 0,)	(1, 0, 0,)	(0, 1, 0,)	(1, 1, 0,)	(0, 1, 1,)

Шаг 4. Результаты вычислений и целевая функция $g_4(t)$ представлены на рис. 7.

Для построения функции $g_4(t)$ необходимо рассмотреть интервалы, образованные точками 0, 2, 3, 5, 8, 0 + 7, 2 + 7. Точки 3 + 7, 5 + 7, 8 + 7 не рассматриваются. Таким образом, достаточно рассмотреть 5 точек. В результате получим:

t	0	2	3	5	8
g	0	5	7	12	13
$x(t)$	(0, 0, 0, 0)	(1, 0, 0, 0)	(0, 1, 0, 0)	(1, 1, 0, 0)	(0, 1, 1, 0)

3.3. Эффективность графического алгоритма

В процессе работы алгоритма было просмотрено всего 17 точек: на первом шаге 2 точки, на втором – 3 точки, на третьем – 7 точек, на четвертом шаге 5 точек. В алгоритме динамического программирования пришлось бы просмотреть $4 \cdot 9 = 36$ точек. Следовательно, для данного примера мы значительно сокращаем псевдополиномиальную составляющую трудоемкости алгоритма.

Стоит отметить, что графический алгоритм работает и в случае, когда $a_i \notin Z$, $\forall i$, $A \notin Z$.

На шаге 3 и 4 видно, что удвоение количества “хранимых” интервалов не происходит. Стоит предположить, что для большого числа примеров трудоемкость графического алгоритма будет полиномиальной.

Чтобы на шаге α появлялось как можно меньше новых “хранимых” интервалов необходимо упорядочить исходные данные в порядке $\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \dots \geq \frac{c_n}{a_n}$. Тогда функция $g^2(t)$ будет “поглощаться” эффективнее.

Алгоритм косвенно учитывает особенности задачи. В алгоритме динамического программирования никак не учитывается, что предмет 4 в приведенном примере наименее предпочтителен (см. $\frac{c_4}{a_4}$). В графическом алгоритме на шаге 4 заметно, что функция $g^2(t)$ не оказала влияния на $g_4(t)$, т.е. была учтена некоторая эвристическая особенность задачи.

4. Заключение

Идея графического подхода является естественным продолжением метода динамического программирования. Данный подход может быть применен к задачам как с нецелочисленными, так и с отрицательными значениями параметров.

Необходимо также отметить, что решение примера из [5, стр. 289]

$$(4) \quad \begin{cases} \sum_{i=1}^n c_i x_i \longrightarrow \max \\ \sum_{i=1}^n 2x_i \leq 2 \left\lfloor \frac{n}{2} \right\rfloor + 1, \end{cases}$$

когда $c_i = 2$, $i = 1, \dots, n$, при $n \geq 4$, может быть найдено с помощью графического подхода за $O(n)$ операций. В общем случае, при произвольных c_i , $i = 1, \dots, n$, пример (4) будет решен за $O(n \log n)$ операций предлагаемым алгоритмом. Алгоритмы, основанные на методе ветвей и границ, находят решение данного примера за $O(2^{\frac{n}{2}})$ операций.

В настоящее время проводится экспериментальное исследование предлагаемого подхода на тестовых примерах “большой” размерности, также будет проведен сравнительный анализ с алгоритмами, основанными на методе ветвей и границ (количества точек ветвления в дереве поиска с количеством “храняемых” интервалов).

В заключение хотелось бы поблагодарить Гафарова Е.Р. за полезное обсуждение полученных результатов.

СПИСОК ЛИТЕРАТУРЫ

1. Пападимитриу Х., Стайглиц К. Комбинаторная оптимизация. Алгоритмы и сложность. М.: Мир, 1985.
2. Keller H., Pferschy U., Pisinger D. Knapsack problems. Springer, 2004.
3. Беллман Р. Динамическое программирование. М.: ИЛ, 1960.
4. Сигал И.Х., Иванова А.П. Введение в прикладное дискретное программирование. М.: Физматлит, 2002.
5. Современное состояние теории исследования операций. Под ред. Н.Н. Моисеева. М.: Наука, 1979.

Статья представлена к публикации членом редколлегии П.Ю. Чеботаревым.

Поступила в редакцию 17.07.2006