Автоматика и телемеханика, № 12, 2016

© 2016 г. Е.Р. ГАФАРОВ, канд. физ.-мат. наук (axel73@mail.ru)
(Институт проблем управления им. В.А. Трапезникова РАН, Москва)

ГРАФИЧЕСКИЙ МЕТОД РЕШЕНИЯ ЗАДАЧ КОМБИНАТОРНОЙ ОПТИМИЗАЦИИ[1]

Предлагается графический метод решения задач комбинаторной оптимизации, которые допускают декомпозицию и использование принципа оптимальности Беллмана при их решении. В отличие от алгоритмов динамического программирования, использующих тот же принцип, в графическом алгоритме все возможные состояния системы рассматриваются не отдельно, а группами. Это становится возможным, если принимать во внимание аналитический вид целевой функции, т.е. работать с "графиком" функции, преобразовывая его на каждой стадии аналитически. Графический метод позволяет значительно сократить трудоемкость решения некоторых задач и строить эффективные аппроксимационные схемы. Результаты численных экспериментов свидетельствуют об эффективности графического метода.

1. Введение

Графический метод является модификацией метода динамического программирования, т.е. использует тот же декомпозиционный подход к решению задач комбинаторной оптимизации.

Словосочетание "динамическое программирование" впервые было использовано в 1940-х гг. Р. Беллманом [1] для описания процесса нахождения решения задачи, где ответ на одну задачу может быть получен только после решения задачи, "предшествующей" ей. Идея метода динамического программирования заключается в следующем. Задачу разбивают на подзадачи, решают их, используя декомпозицию, и потом объединяют решения подзадач в одно общее решение. В процессе декомпозиции возникает множество идентичных подзадач, из которых решается только одна, что позволяет сократить объем вычислений.

Наряду с методом ветвей и границ, метод динамического программирования чаще всего используется для решения задач комбинаторной оптимизации.

Известно [2], что для Задачи о ранце все алгоритмы ветвей и границ с полиномиальными алгоритмами расчета верхних и нижних оценок имеют трудоемкость не меньше $\frac{3}{2}\frac{2^{n+3/2}}{\sqrt{\pi(n+1)}}$, т.е. $2^{O(x)}$ операций, где n — количество предметов, а x — длина входа. По сути, такие алгоритмы ветвей и границ не сильно отличаются по трудоемкости от полного перебора, имеющего экспоненциальную трудоемкость. Аналогичные результаты получены и для других задач.

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (проекты №№ 13-01-12108, 13-08-13190, 15-07-03141, 15-07-07489).

В 2010 г. О'Нил в [3] показал, что широко известная модификация метода динамического программирования позволяет решать многие классические задачи за субэкспоненциальное время $2^{O(\sqrt{x})}$.

Таким образом, на данный момент с теоретической точки зрения метод динамического программирования является более перспективным по сравнению с методом ветвей и границ и другими точными методами решения, и его развитие является оправданным. Графический метод, идея которого впервые представлена в [4], является одним из возможных направлений развития.

Будем обозначать алгоритм динамического программирования через DPA (Dynamic Programming Algorithm), а графический алгоритм — GrA (Graphical Algorithm). В DPA на каждом шаге (на каждой стадии) j вычисляются значения некоторой функции $F_j(t)$ для каждого возможного значения аргумента t (для каждого состояния) процесса принятия решения, где $t \in [0, C]$ и $t \in Z$. Будем называть функцию $F_j(t)$ функцией Беллмана. Фактически эта функция соответствует значениям целевой функции для подзадачи размерности j, если она решается в условиях (при состоянии системы) t. На практике DPA реализуется в режиме "калькулятора", т.е. перебираются все состояния $t \in [0, C]$ и $t \in Z$, для каждого состояния проводятся несложные вычисления, и полученное значение $F_j(t)$ записывается в ячейку памяти (в таблицу). Эти C числовых значений используются на следующем шаге $j + 1$. Однако часто отпадает необходимость в вычислении (и сохранении в памяти) значения $F_j(t)$ для каждой точки t. Может оказаться, что на некотором интервале $[t_l, t_{l+1})$ вычисляемая функция представима аналитически в виде $F_j(t) = \varphi(t)$ (например, $F_j(t) = k \cdot t + b$, т.е. функция $F_j(t)$ определена в том числе для нецелых значений аргумента t). Тогда процесс вычисления функции $F_{j+1}(t)$ на шаге $j + 1$ можно организовать таким образом, чтобы учитывать не отдельные значения $F_j(t)$, а преобразовывать функцию $F_j(t)$ в $F_{j+1}(t)$ аналитически, согласно заданным рекурсивным уравнениям Беллмана.

Пусть для некоторой проблемы минимизации целевого функционала заданы рекурсивные уравнения Беллмана:

$$F_j(t) = \min \begin{cases} \Phi^1(t) = \alpha_j^1(t) + F_{j-1}(t - a_j^1), & j = 1, 2, \ldots, n, \\ \Phi^2(t) = \alpha_j^2(t) + F_{j-1}(t - a_j^2), & j = 1, 2, \ldots, n, \\ \ldots & \ldots \\ \Phi^{k_j}(t) = \alpha_j^{k_j}(t) + F_{j-1}(t - a_j^{k_j}), & j = 1, 2, \ldots, n, \end{cases} \tag{1}$$

с начальными условиями:

$$\begin{aligned} F_0(t) &= 0, \quad \text{для} \quad t \geqslant 0, \\ F_0(t) &= +\infty, \quad \text{для} \quad t < 0. \end{aligned} \tag{2}$$

В (1) значения $a_j^k,\ k = 1, \ldots, k_j$, будем называть "стратегиями", определенными для данного элемента системы j. Каждой стратегии $a_j^k,\ k = 1, \ldots, k_j$, соответствует функция $\Phi_j(t)$ и значение переменной $x_j = x_j^k$, участвующей в итоговом решении. Переменная x_j – управляемый параметр на шаге j процесса. Трудоемкость DPA для такой системы уравнений равна $O(\sum_{j=1}^{n} k_j C)$.

Таблица 1. Вычисления в DPA

t	0	1	2	...	y	...	C
$F_j(t)$	$value_0$	$value_1$	$value_2$	...	$value_y$	...	$value_C$
частичное оптимальное решение $X(t)$	$X(0)$	$X(1)$	$X(2)$	...	$X(y)$	...	$X(C)$

Таблица 2. Вычисления в GrA

t	$[t_0, t_1)$	$[t_1, t_2)$	...	$[t_l, t_{l+1})$	...	$[t_{m_j-1}, t_{m_j}]$
$F_j(t)$	$\varphi_1(t)$	$\varphi_2(t)$	...	$\varphi_{l+1}(t)$	...	$\varphi_{m_j}(t)$
частичное оптимальное решение $X(t)$	$X(t_0)$	$X(t_1)$	...	$X(t_l)$	...	$X(t_{m_j-1})$

Для задач с булевыми переменными, таких как Задача о ранце или Задача разбиения, для некоторых одноприборных задач теории расписаний [4–6] выполняется $k_j = 2$, $j = 1, \dots, n$ и $x_j^1 = 1$, $x_j^2 = 0$. Если в таких задачах вычисление DPA необходимо провести на каждом шаге $j = 1, 2, \dots, n$, где n размерность задачи, то трудоемкость DPA обычно равна $O(nC)$. Переменная x_j — для Задачи о ранце имеет интерпретацию "включать или не включать предмет в ранец". На шаге j, $j = 1, 2, \dots, n$, DPA вычисляются и сохраняются в памяти компьютера данные, представленные в табл. 1.

В табл. 1 $X(y)$, $y = 0, 1, \dots, C$, — вектор, описывающий частичное оптимальное решение, который состоит из j элементов (значений) $x_1, x_2, \dots, x_j$. Та же самая информация иногда может быть представлена в виде, приведенном в табл. 2, где $0 = t_0 < t_1 < t_2 < \dots < t_{m_j} = C$. Точки $t_1, \dots, t_{m_j}$ будем называть *точками излома*, так как в них изменяется уравнение, задающее функцию $F_j(t)$.

Для вычисления функции $F_{j+1}(t)$ в GrA необходимо сравнить промежуточные функции $\Phi^1(t), \dots, \Phi^{k_{j+1}}(t)$, которые определяются следующим образом.

Функция $\Phi^k(t)$ является комбинацией двух функций $\alpha_{j+1}^k(t)$ и $F_j(t - a_{j+1}^k)$. Функция $F_j(t - a_{j+1}^k)$ имеет ту же структуру, что и в табл. 2, но где каждый интервал $[t_l, t_{l+1})$ заменен на интервал $[t_l - a_{j+1}, t_{l+1} - a_{j+1}^k)$, что означает смещение графика функции $F_j(t)$ вправо на величину a_{j+1}^k. Если функцию $\alpha_{j+1}^k(t)$ можно представить в той же форме, что и в табл. 2 с μ_k столбцами, то можно сохранить функцию $\Phi^k(t)$ в памяти компьютера в той же форме, что и в табл. 2, но с $m_j + \mu_k$ столбцами. Здесь μ_k – количество интервалов, на каждом из которых функция $\alpha_{j+1}^k(t)$, $k = 1, \dots, k_{j+1}$, задается некоторым фиксированным уравнением от t.

Функция $F_{j+1}(t) = \min\{\Phi^1(t), \dots, \Phi^{k_{j+1}}(t)\}$ вычисляется следующим образом. Пусть таблица, которой задана функция $\Phi^k(t)$, содержит интервалы (столбцы)

$$[t_0^k, t_1^k),\ [t_1^k, t_2^k),] \dots,][t_{(m_j+\mu_k)-1}^k, t_{(m_j+\mu_k)}^k], \quad k = 1, \dots, k_j.$$

Для вычисления функции $F_{j+1}(t)$ необходимо сравнить все функции $\Phi^k(t)$, $k = 1, \ldots, k_j$, на каждом интервале, сформированном точками

$$\left\{t_0^k, t_1^k, t_2^1, \ldots, t^k_{(m_j+\mu_k)-1}, t^k_{(m_j+\mu_1)} | k = 1, \ldots, k_j\right\}.$$

На каждом из этих интервалов уравнение, задающее функцию $\Phi^k(t)$, $k = 1, \ldots, k_j$, неизменно, поэтому можно аналитически найти точки их пересечения и построить функцию минимума. Пусть μ' – общее количество точек пересечения. Тогда в таблице, соответствующей функции $F_{j+1}(t)$, может быть $M = k_{j+1}m_j + \sum_{i=1}^{k_{j+1}} \mu_i + \mu'$ интервалов. GrA можно построить таким образом, что все нецелочисленные точки излома будут исключены из таблицы $F_{j+1}(t)$. Например, если в таблице присутствуют два интервала $[t_{k-1}, t_k)$ и $[t_k, t_{k+1})$, где $t_k \notin Z$, то можно заменить интервалы на $[t_{k-1}, \lfloor t_k \rfloor]$ и $[\lfloor t_k \rfloor + 1, t_{k+1})$ и учитывать полученный разрыв на последующих шагах. Тогда $M \leqslant C$. Фактически на каждом шаге j, $j = 1, 2, \ldots, n$, графического алгоритма не рассматриваются все точки $t \in [0, C]$, $t \in Z$, а рассматриваются только те точки, в которых меняется оптимальное частичное решение или меняется аналитический вид функции $F_j(t)$. Для некоторых задач (для некоторых целевых функций), количество таких точек M небольшое, поэтому графический алгоритм имеет трудоемкость $O(\sum_{j=1}^n k_j \min\{C, M\})$ операций вместо $O(\sum_{j=1}^n k_j C)$ операций, как в алгоритме динамического программирования.

Более того, GrA имеет следующие преимущества перед DPA:

- с помощью GrA можно решать примеры с нецелочисленными параметрами;
- время работы GrA в двух примерах с множеством числовых параметров P и множеством параметров $\{b \cdot 10^l \pm 1 | b \in P\}, k > 1$, совпадает, в то время как время работы DPA будет в 10^l раз больше во втором примере, т.е. с помощью GrA, можно решать примеры “большого масштаба”;
- принимаются во внимание внутренние свойства задачи (например, для Задачи о ранце, может оказаться, что предмет с наименьшей удельной стоимостью не оказывает влияния на функцию Беллмана);
- известно, что GrA для некоторых задач имеет полиномиальную трудоемкость, в то время как исходный алгоритм DPA – псевдополиномиальную, или GrA существенно сокращает трудоемкость DPA [5].

Необходимо отметить, что для некоторых примеров рассмотренных задач трудоемкость GrA равна трудоемкости DPA. Можно привести примеры задач с нецелочисленными параметрами, для которых трудоемкость GrA является экспоненциальной [4].

В последующих разделах представлены более детальные описания GrA для некоторых одноприборных задач теории расписаний и для задачи об инвестициях.

2. Графический метод решения некоторых задач теории расписаний

Рассматриваемые в данном разделе задачи формулируются следующим образом. На одном приборе необходимо обслужить множество $N = \{1, 2, \ldots$

$\ldots, n\}$ из n требований. Прерывания в обслуживании требований не допускаются. Прибор обслуживает одновременно не более одного требования. Все требования поступают на обслуживание в нулевой момент времени. Для каждого требования $j \in N$ заданы продолжительность обслуживания $p_j > 0$, вес $w_j > 0$ и директивный срок $d_j > 0$, к которому в идеале требование должно быть обслужено.

Допустимое решение представимо в виде перестановки $\pi = (j_1, j_2, \ldots, j_n)$ требований из множества N. Перестановка задает порядок обслуживания требований на приборе. Пусть $C_{j_k}(\pi) = \sum_{l=1}^{k} p_{j_l}$ — время завершения обслуживания требования j_k в расписании, полученном из перестановки π. Если $C_j(\pi) > d_j$, тогда требование j запаздывает, если же $C_j(\pi) \leqslant d_j$, то требование j не запаздывает. Пусть $T_j(\pi) = \max\{0, C_j(\pi) - d_j\}$ — запаздывание требования j при расписании, полученном из перестановки π, и пусть $GT_j(\pi) = \min\{\max\{0, C_j(\pi) - d_j\}, p_j\}$.

Для задачи минимизации взвешенного запаздывания $1||\sum w_j T_j$ необходимо найти оптимальную перестановку (расписание) π^*, которая минимизирует функцию $F(\pi) = \sum_{j=1}^{n} w_j T_j(\pi)$. Аналогично для задачи минимизации суммарного запаздывания $1||\sum T_j$ необходимо минимизировать $F(\pi) = \sum_{j=1}^{n} T_j(\pi)$. Для обобщенной задачи запаздывания $1||\sum GT_j$ целевая функция $F(\pi) = \sum_{j=1}^{n} GT_j(\pi)$. Рассматриваются также следующие задачи и частные случаи:

- минимизация взвешенного суммарного запаздывания при одинаковом директивном сроке $d_j = d,\ j = 1, 2, \ldots, n$. Обозначим данную проблему $1|d_j = d|\sum w_j T_j$;
- специальный случай $B-1$ задачи минимизации суммарного запаздывания $1||\sum T_j$, где $p_1 \geqslant p_2 \geqslant \ldots \geqslant p_n,\ d_1 \leqslant d_2 \leqslant \ldots \leqslant d_n$ и $d_n - d_1 \leqslant p_n$;
- специальный случай $B-1G$ задачи $1||\sum T_j$, где $d_{\max} - d_{\min} \leqslant p_{\min}$, $d_{\max} = \max_{j\in N}\{d_j\}$, $d_{\min} = \min_{j\in N}\{d_j\}$ и $p_{\min} = \min_{j\in N}\{p_j\}$;
- задача максимизации взвешенного суммарного запаздывания $1(no - idle)||\max\sum w_j T_j$, в которой при любом допустимом расписании требования обслуживаются без простоев с нулевого момента времени;
- частный случай предыдущей задачи, где $w_j = 1,\ j = 1, 2 \ldots, n$, обозначаемый $1(no - idle)||\max\sum T_j$.

Все перечисленные задачи и частные случаи (кроме задачи $1(no - idle)||\max\sum T_j$) являются NP-трудными в обычном смысле. Ссылки на результаты об их сложности представлены, например, в [6, 7].

Для частного случая $B-1$ могут быть выписаны следующие уравнения Беллмана [7]:

(3)
$$F_j(t) = \min \begin{cases} \Phi^1(t) = \max\{0, t + p_j - d_j\} + F_{j-1}(t + p_j), & j = 1, 2, \ldots, n, \\ \Phi^2(t) = \max\left\{0, t + \sum\limits_{i=1}^{j} p_i - d_j\right\} + F_{j-1}(t), & j = 1, 2, \ldots, n. \end{cases}$$

Таблица 3. Функция $F_j(t)$ в GrA для случая $B-1$

k	1	2	$\ldots$	m_j+1
интервал k	$(-\infty, t_j^1]$	$(t_j^1, t_j^2]$	$\ldots$	$(t_j^{m_j}, +\infty)$
b_j^k	0	b_j^2	$\ldots$	$b_j^{m_j+1}$
u_j^k	0	u_j^2	$\ldots$	$u_j^{m_j+1}$
π_j^k	π_j^1	π_j^2	$\ldots$	$\pi_j^{m_j+1}$

Начальные условия те же. При этом требования перенумерованы согласно правилу $p_1 \leqslant p_2 \leqslant \ldots \leqslant p_n$. Состояние системы t означает момент времени, с которого начнется обслуживание подмножества требований $\{1, \ldots, j\}$. Функции Φ_1 соответствует выбор $x_j = 0$, т.е. требование j ставится в конец частичного расписания (перестановки из требований подмножества $\{1, \ldots, j\}$), а функции Φ_2 – выбор $x_j = 1$, т.е. требование j ставится в начало расписания.

Для всех вышеупомянутых задач функции Беллмана являются кусочно-линейными непрерывными возрастающими функциями, т.е. на каждом интервале функцию Беллмана можно описать с помощью двух числовых параметров — наклон прямой и значение функции в начале интервала. Тогда функции $F_j(t)$ можно хранить в виде, показанном в табл. 3.

Записи в табл. 3 означают следующее. Для каждого значения $t \in (t_j^{k-1}, t_j^k]$ оптимальным будет частичное расписание π_j^k, в котором u_j^k запаздывающих требований (наклон функции) и которому соответствует значение функции Беллмана $F_j(t) = u_j^k \cdot (t - t_j^{k-1}) + b_j^k$. При этом $b_j^1 < b_j^2 < \ldots < b_j^{m_j+1}$.

Чтобы построить функцию $\Phi^1(t)$, нужно сместить график функции $F_{j-1}(t)$ влево на p_j единиц и вычислить точку (состояние) $t' = d_j - p_j$, начиная с которой требование j начинает запаздывать. Поместить эту точку в таблицу смещенной функции $F_{j-1}(t)$ и для всех столбцов, где $t > t'$, увеличить наклон на единицу. Чтобы построить функцию $\Phi^2(t)$, нужно вычислить точку (состояние) $t'' = d_j - \sum_{i=1}^{j} p_i$, начиная с которой требование j начинает запаздывать. Поместить эту точку в таблицу функции $F_{j-1}(t)$ и для всех столбцов, где $t > t''$, увеличить наклон на единицу. Соответственно чтобы найти $F_j(t) = \min\{\Phi^1(t), \Phi^2(t)\}$, придется рассмотреть не более $2(m_j + 1) + 2$ интервалов.

Оба алгоритма, GrA и DPA, основанные на данном принципе оптимальности, имеют одинаковую трудоемкость $O(nd_{\max}), d_{\max} = \max_{j\in N} d_j$, так как для всех состояний системы $t > d_{\max}$, $x_j = 1$ [7].

Для задачи $1(no-idle)|| \max \sum w_j T_j$ могут быть выписаны следующие уравнения Беллмана [7]:

$$(4) \quad F_j(t) = \max \begin{cases} \Phi^1(t) = w_j \max\{0, t + p_j - d_j\} + F_{j-1}(t + p_j), & j = 1, 2, \ldots, n, \\ \Phi^2(t) = w_j \max\left\{0, t + \sum\limits_{i=1}^{j} p_i - d_j\right\} + F_{j-1}(t), & j = 1, 2, \ldots, n. \end{cases}$$

Так как функции $\alpha_j(t)$ являются выпуклыми и решается задача максимизации, т.е. $F_j(t) = \max\{\Phi^1(t), \Phi^2(t)\}$, то функция $F_j(t)$ также является выпуклой. Преобразования функции $F_j(t)$ в функции $\Phi^1(t)$ и $\Phi^2(t)$ выполняются аналогичным образом. Наклон при этом меняется не на единицу, а на значение w_j. Тогда функция $F_j(t)$ задается не более $\sum_{i=1}^{j} w_i$ столбцами [7] (так как функция возрастающая и выпуклая). Тогда трудоемкость GrA будет $O(n \min\{d_{\max}, \sum w_j\})$, что меньше трудоемкости DPA $O(nd_{\max})$. Более того, для частного случая $1(no-idle)||\max \sum T_j$ выполняется равенство $\sum w_j = n$ и GrA является полиномиальным в отличие от псевдо-полиномиального DPA.

Таким образом, с помощью GrA можно снизить трудоемкость решения некоторых одноприборных задач теории расписаний.

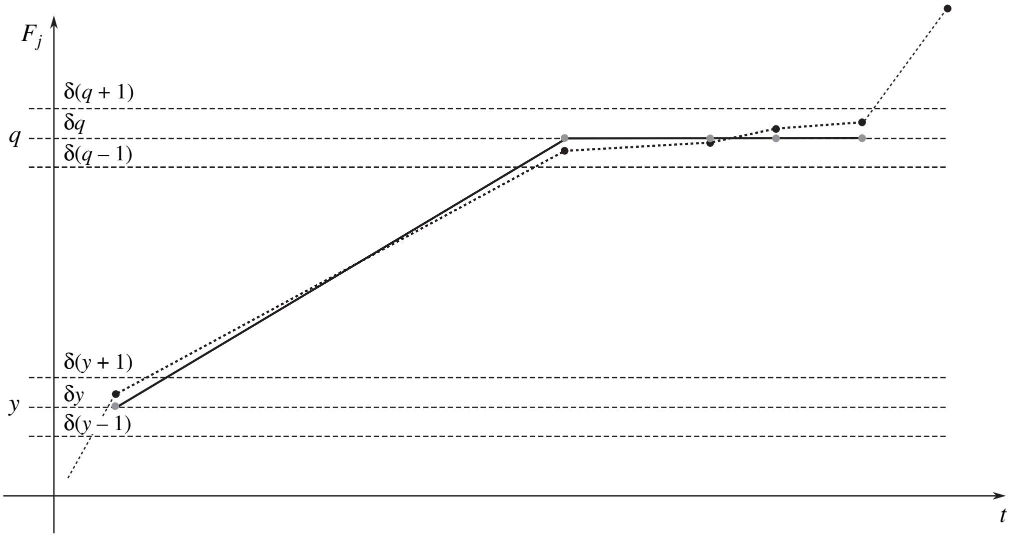
3. Аппроксимационные схемы решения, основанные на графическом методе

На основе графического метода легко построить FPTAS (fully-polynomial time approximation schema). Полиномиальный алгоритм, который для задач теории расписаний минимизации функции $F(\pi)$ находит решение π' такое, что $F(\pi')$ не более чем в $\rho \geqslant 1$ раз больше $F(\pi^*)$, называется *ρ-аппроксимационным алгоритмом. Значение ρ называется максимальной погрешностью.* Если для задачи существует ρ-аппроксимационный алгоритм, то говорят, что задача аппроксимируема с относительной погрешностью ρ. Набор ρ-аппроксимационных алгоритмов называется полностью полиномиальной аппроксимацонной схемой (fully polynomial-time approximation scheme – FPTAS), если $\rho = 1 + \varepsilon$ для любого $\varepsilon > 0$ и трудоемкость алгоритма полиномиально зависит только от размерности задачи и значения $1/\varepsilon$. Напомним, что проблемы NP-трудные в сильном смысле не имеют FPTAS, если P $\neq$ NP.

Графический алгоритм может быть модифицирован в FPTAS следующим образом. Пусть рассматривается некоторая задача минимизации. Пусть $\delta =$

Таблица 4. Сравнение трудоемкостей GrA, DPA и FPTAS, основанной на GrA [7]

Задача	Трудоемкость GrA	Трудоемкость FPTAS	Трудоемкость классического DPA
$1\|\|\sum w_j U_j$ [7]	$O(\min\{2^n, n \cdot \min\{d_{\max}, F_{opt}\}\})$	–	$O(nd_{\max})$
$1\|d_j = d'_j + A\|\sum U_j$ [7]	$O(n^2)$	–	$O(n\sum p_j)$
$1\|\|\sum GT_j$	$O(\min\{2^n, n \cdot \{d_{\max}, nF^*\}\})$	$O(n^2 \log\log n + \frac{n^2}{\varepsilon})$	$O(nd_{\max})$
$1\|\|\sum T_j$ частный случай $B-1$	$O(\min\{2^n, n \cdot \min\{d_{\max}, F^*\}\})$	$O(n^2/\varepsilon)$	$O(nd_{\max})$
$1\|\|\sum T_j$ частный случай $B-1G$	$O(\min\{2^n, n^2 \cdot \min\{d_{\max}, F^*\}\})$	$O(n^3/\varepsilon)$	$O(n^2 d_{\max})$
$1\|d_j = d\|\sum w_j T_j$	$O(\min\{2^n, n^2 \cdot \min\{d, F^*\}\})$	$O(n^3/\varepsilon)$	$O(n^2 d_{\max})$
$1(no-idle)\|\|\max\sum w_j T_j$	$O(\min\{2^n, n \cdot \min\{d_{\max}, nF^*, \sum w_j\}\})$ [7]	$O(n^2 \log\log n + \frac{n^2}{\varepsilon})$	$O(nd_{\max})$
$1(no-idle)\|\|\max\sum T_j$	$O(n^2)$ [5]	–	$O(nd_{\max})$



Аппроксимация функции $F_j(t)$.

$= \varepsilon LB/2n$, где LB – некоторая известная нижняя оценка целевой функции. Пусть также известна верхняя оценка UB такая, что UB/LB не превышает некоторой константы c. Чтобы сократить трудоемкость GrA, необходимо сократить количество столбцов в таблицах $F_j(t)$, а количество этих столбцов равно количеству различных значений $0 = b_j^1, b_j^2, b_j^3, \ldots, b_j^{m_j+1}$ в таких столбцах. Можем рассматривать только те состояния системы (и интервалы), для которых $F_j(t) < UB$, так как остальные значения не влияют на целевую функцию. Тогда примем $b_j^{m_j+1} < UB$.

В таблице $F_j(t)$ будем хранить не оригинальные значения b_j^k, а значения $\overline{b_j^k}$, являющиеся ближайшими к b_j^k значениями, делящимися без остатка на δ. Существует не более чем $UB/\delta = cn/\varepsilon$ различных значений $\overline{b_j^k}$. Тогда можно будет преобразовать таблицу функции $F_j(t)$ в таблицу приближенной функции с не более чем $2cn/\varepsilon$ столбцами. Причем для данной модифицированной функции $F'(t)$ выполняется $|F(t) - F'(t)| < \delta \leqslant \varepsilon F(\pi^*)/n$. Если будем выполнять подобную аппроксимацию после каждой стадии GrA, то накопленная ошибка не превысит значения $n\delta \leqslant \varepsilon F(\pi^*)$ и трудоемкость GrA будет $O(\frac{n^2}{\varepsilon})$. Таким образом, будет получена FPTAS.

Подробное описание данной аппроксимации имеется в [7]. На рисунке схематично представлена аппроксимация функции $F_j(t)$. В табл. 4 представлено сравнение трудоемкостей GrA, DPA и FPTAS, основанной на GrA для некоторых однориборных задач теории расписаний.

4. Графический алгоритм решения задачи распределения инвестиций

В данном разделе представлен графический алгоритм решения задачи распределения инвестиций. В задаче даны множество N из n потенциальных инвестиционных проектов и доступный объем инвестиций $A > 0$. Для каждого проекта $j,\ j=1,\dots,n$, задана функция прибыли $f_j(t),\ t\in[0,A]$. Значение $f_j(t')$ означает прибыль, полученную от проекта j, если в него инвестировать t' денежных единиц. Необходимо определить объемы инвестиций $\tau_j\in[0,A], \tau_j\in Z$, для каждого проекта $j\in N$ такие, чтобы $\sum_{j=1}^{n}\tau_j\leqslant A$ и значение $\sum_{j=1}^{n}f_j(\tau_j)$ было максимальным. Будем считать, что все функции $f_j(t),\ j=1,\dots,n$, являются неубывающими кусочно-линейными.

Данная задача может быть решена с помощью DPA [8] с использованием следующих уравнений Беллмана:

$$F_j(T)=\max_{t=0,1,\dots,T}\{f_j(t)+F_{j-1}(T-t)\},\quad T=A,A-1,\dots,1,\quad j=1,\dots,n.$$

Трудоемкость DPA – $O(nA^2)$ операций. Очевидно, что функции $f_j(t),\ j=1,\dots,n$, представимы в табличном виде с тремя строками, по аналогии с функциями $F_j(t)$ в разделе 3. Трудоемкость альтернативного DPA — $O(\sum kA)$, где $\sum k$ – количество кусочно-линейных фрагментов функций $f_j(t),\ j=1,\dots,n$.

Идея графического алгоритма заключается в следующем. Чтобы найти значение $F_j(A)$, необходимо на плоскости изобразить график функции $F_j(A)$, начинающийся в точке ноль, и график зеркальной функции $f'_j(t)$, перевернутый относительно оси F и смещенный вправо из точки ноль в точку A. Тогда несложно построить график функции максимума F' из функций $F_j(A)$ и $f'_j(t)$ и найти максимум данной функции на интервале $[0,A]$. Так как обе функции являются кусочно-линейными, то максимум достигается в точках t, соответствующих точкам излома двух функций. Для последующих состояний T системы также будут рассматриваться только точки, соответствующие точкам излома. Для нахождения $F_j(A-\varepsilon)$ необходимо сместить график $f'_j(t)$ влево на ε. Так как обе функции кусочно-линейные, то несложно задать каждой точке Υ излома графиков некоторое уравнение $-u_\Upsilon\cdot\varepsilon+b_\Upsilon$, характеризующее изменение значения функции максимума F' при смещении графика $f'_j(t)$. Необходимо отслеживать точки, в которых эти уравнения меняются или "пересекаются", чтобы определять уравнения, задающие $F_j(T)$ на некотором интервале. Таким образом, можно найти уравнение прямой, задающей значения функции $F_j(T)$ на интервале $[A-\varepsilon,A]$. Подробное описание GrA приведено в [8].

Теорема. *GrA находит оптимальное решение для всех $T\in[0,A],\ T\in Z$, за время $O(\sum kA)$.*

Несмотря на то что трудоемкость GrA не меньше трудоемкости DPA, GrA имеет ряд преимуществ, изложенных в разделе 2. На основе GrA предложена полностью полиномиальная схема приближенного решения задачи с меньшей известной трудоемкостью среди аналогичных алгоритмов $O(\frac{n\cdot\sum k}{\varepsilon}(1+\log\log n))$. Полученное таким алгоритмом приближенное значение целевой функции не более чем в $1+\varepsilon$ раз меньше оптимального значения.

5. Результаты экспериментальных исследований

Для некоторых одноприборных задач теории расписаний были проведены численные эксперименты с целью определения трудоемкости GrA для примеров, параметры которых распределены по равномерному закону. Проводились следующие эксперименты.

Для задачи минимизации взвешенного количества запаздывающих требований для одного прибора $1||\sum w_j U_j$ [6] генерировались 2500 примеров для каждой размерности задачи $n \in \{4, 5, \ldots, 50\}$, где $p_j \in [0, 100]$, $w_j \in [1, 100]$, директивные сроки в интервале $[0, \sum p_j]$. Для каждого примера подсчитывалось количество точек излома в алгоритме GrA на всех стадиях. По результатам экспериментов суммарное количество точек излома не превышает n^2 для всех рассмотренных примеров, т.е. трудоемкость алгоритма для рассмотренных примеров не превосходит $O(n^3)$ операций.

Для задачи $1(no-idle)||\max\sum w_j T_j$ проводились аналогичные эксперименты. Для каждой размерности задачи $n \in \{20, 40, 60, 80, 100, 150, 200, \ldots \ldots, 1000\}$ генерировались 10000 примеров, где $p_j \in [0, 100]$, $w_j \in [1, n]$, директивные сроки в интервале $[0, \sum p_j]$. Для рассмотренных примеров среднее суммарное количество точек излома не превышает n^2. Максимальное количество точек излома для рассмотренных примеров меньше $O(n^3)$.

6. Заключение

Графический метод можно эффективно применять для решения задач комбинаторной оптимизации, для которых существует псевдополиномиальный алгоритм решения, основанный на принципе оптимальности Беллмана. Для многих одноприборных задач теории расписаний и для известных комбинаторных задач (Задача о ранце, Задача распределения инвестиций) графический метод позволяет существенно сократить трудоемкость по сравнению с классическими алгоритмами динамического программирования. Графический метод также позволяет генерировать аппроксимационные схемы с лучшей трудоемкостью среди известных псевдополиномиальных алгоритмов. Результаты экспериментальных исследований свидетельствуют об эффективности метода. Таким образом, графический метод имеет как теоретическое, так и практическое значения.

СПИСОК ЛИТЕРАТУРЫ

1. *Bellman R.* Dynamic Programming, Princeton: Princeton Univ. Press, 1957.
2. *Posypkin M.A., Sigal I.Kh.* Speedup estimates for some variants of the parallel implementations of the branch-and-bound method // J. Math. Math. Physics. 2006. V. 46. No. 12. P. 2189–2202.
3. *O'Neil E.T., Kerlin S.* A Simple $2^{O(\sqrt{x})}$ Algorithm for PARTITION and SUBSET SUM. 2010. http://www.lidi.info.unlp.edu.ar/WorldComp2011-Mirror/FCS8171.pdf
4. *Lazarev A.A., Werner F.* A Graphical Realization of the Dynamic Programming Method for Solving NP-Hard Combinatorial Problems // Comput. Math. Appl. 2009. V. 58. No. 4. P. 619–631.
5. *Gafarov E.R., Lazarev A.A., Werner F.* Transforming a Pseudo-Polynomial Algorithm for the Single Machine Total Tardiness Maximization Problem into a Polynomial One // Ann. Oper. Res. 2012. V. 196. No. 1. P. 247–261.
6. *Gafarov E.R., Lazarev A.A., Werner F.* A Note on a Single Machine Scheduling Problem with Generalized Total Tardiness Objective Function // Inform. Process. Lett. 2012. V. 112. No. 3. P. 72–76.

7. *Gafarov E.R., Dolgui A., Werner F.* A New Graphical Approach for Solving Single Machine Scheduling Problems Approximately // Int. J. Product. Res. 2014. V. 52. No. 13. P. 3762–3777.

8. *Гафаров Е.Р., Долгий А.Б., Лазарев А.А. и др.* Модификация метода динамического программирования для решения задачи об инвестициях // АиТ. 2016. № 9. С. 150–166.

Статья представлена к публикации членом редколлегии А.А. Лазаревым.

Поступила в редакцию 04.02.2016