ISSN 0005-1179, Automation and Remote Control, 2007, Vol. 68, No. 4, pp. 583–592. © Pleiades Publishing, Ltd., 2007.
Original Russian Text © A.A. Lazarev, 2007, published in Avtomatika i Telemekhanika, 2007, No. 4, pp. 13–23.

Graphic Approach to Combinatorial Optimization[1]

A. A. Lazarev

Dorodnitsyn Computer Center, Russian Academy of Sciences, Moscow, Russia

Received July 17, 2006

Abstract—Consideration was given to a graphic realization of the method of dynamic programming. Its concept was demonstrated by the examples of the partition and knapsack problems. The proposed method was compared with the existing algorithms to solve these problems.

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

DOI: 10.1134/S0005117907040029

1. INTRODUCTION

The classical partition and knapsack problems of combinatorial optimization that are NP-hard in the ordinary sense were considered in [1, 2]. We discuss here their following formulations.

Partition problem. Given is an ordered set $B = \{b_1, \dots, b_n\}$, $b_1 \geq \dots \geq b_n$ of n positive integers. Needed is to partition the set B into two subsets B_1 and B_2 so as to minimize

$$\left| \sum_{b_i \in B_1} b_i - \sum_{b_i \in B_2} b_i \right| \longrightarrow \min. \tag{1}$$

One-dimensional knapsack problem is representable as the following problem of integer linear programming:

$$\begin{cases} f(x) = \sum\limits_{i:=1}^{n} c_i x_i \longrightarrow \max \\ \sum\limits_{i:=1}^{n} a_i x_i \leq A \\ x_i \in \{0, 1\},\ i = 1, \dots, n. \end{cases} \tag{2}$$

Problems (1) and (2) are equivalent if $c_i = a_i = b_i$, $i = 1, \dots, n$, and $A = \frac{1}{2} \sum\limits_{j=1}^{n} b_j$.

2. GRAPHIC ALGORITHM FOR THE PARTITION PROBLEM

At steps $\alpha = 1, \dots, n$ the algorithm successively considers the following functions:

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

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

[1] The work was executed within the framework of the scientific school of Acad. Yu.I. Zhuravlev, NSh-5833.2006.1.

In the function $B_1(t)$, as well as in $B_2(t)$, a set of elements $\overline{B_1}$ (correspondingly, $\overline{B_2}$) is associated with each point t from the interval $\left[-\sum\limits_{j:=1}^{\alpha} b_j,\ \sum\limits_{j:=1}^{\alpha} b_j\right]$ under consideration. It deserves noting that at each step α of the algorithm $B_1(t) \bigcup B_2(t) = \{b_1, \ldots, b_{\alpha-1}\} = \overline{B_1} \bigcup \overline{B_2},\ \forall\ t$.

2.1. Concept of the Graphic Algorithm

The current number b_α, where $\alpha = 1, \ldots, n$, is included successively in the set $B_1(t)$ or $B_2(t)$. At each point t, the number b_α is included so as to minimize the function $\left|\sum\limits_{b_j \in B_1(t)} b_j + t - \sum\limits_{b_j \in B_2(t)} b_j\right|$, where $t = t_1 - t_2$ and t_1 and t_2 are added, respectively, to $B_1(t)$ and $B_2(t)$. The parameter t characterizes the sum of numbers that will be added to $B_1(t)$ at the subsequent steps. If $t < 0$, then $-t$ is equal to the sum that will be added to $B_2(t)$ at the subsequent steps. At the current step α, the following function is constructed from the function $F_{\alpha-1}(t) = \left|\sum\limits_{b_j \in B_1(t)} b_j - \sum\limits_{b_j \in B_2(t)} b_j\right|$ obtained at the preceding step $\alpha - 1$:

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

where $F_0(t) := 0,\ \forall\ t$. If $F_\alpha^1(t) < F_\alpha^2(t)$, then $B_1(t) := B_1(t - b_\alpha) \bigcup \{b_\alpha\}$; otherwise, $B_2(t) := B_2(t + b_\alpha) \bigcup \{b_\alpha\}$. The piecewise-linear function $F_\alpha(t)$ may be represented (and stored) in the tabular form at the "break" points: $t_0; t_1; \ldots; t_{m_\alpha}$. Over the interval $\left[t_i - \dfrac{t_i - t_{i-1}}{2}, t_i + \dfrac{t_{i+1} - t_i}{2}\right)$, $i = 1, \ldots, m_\alpha - 1$, the piecewise-linear function $F_\alpha(t)$ obeys the equation $F_\alpha(t) = |t - t_i|$, that is, its graph intersects the axis t at the point t_i. Some fixed partition $(\overline{B}_1; \overline{B}_2)$ corresponds to each time interval $\left[t_i - \dfrac{t_i - t_{i-1}}{2}, t_i + \dfrac{t_{i+1} - t_i}{2}\right)$, that is, $B_1(t^1) = B_1(t^2) = \overline{B}_1$, $\forall t^1$, $t^2 \in \left[t_i - \dfrac{t_i - t_{i-1}}{2}, t_i + \dfrac{t_{i+1} - t_i}{2}\right)$ (similarly for $B_2(t)$). Let a tabular function $F_{\alpha-1}(t)$ be obtained at the preceding step $\alpha - 1$. At the step α, consideration is given to the two functions $F_{\alpha-1}(t - b_\alpha)$ and $F_{\alpha-1}(t + b_\alpha)$ defined, respectively, by two tables at the "break" points: $t_0 - b_\alpha, \ldots t_i - b_\alpha, \ldots, t_{m_{\alpha-1}} - b_\alpha$ and $t_0 + b_\alpha, \ldots, t_i + b_\alpha, \ldots, t_{m_{\alpha-1}} + b_\alpha$.

By comparing the graphs of both functions, we consider the time intervals $[t^1, t^2]$, $t^1, t^2 \in \{t_0 - b_\alpha, \ldots, t_{m_{\alpha-1}} - b_\alpha, t_0 + b_\alpha, \ldots, t_{m_{\alpha-1}} + b_\alpha\}$ where the function $F_{\alpha-1}(t - b_\alpha)$ (and, correspondingly, $F_{\alpha-1}(t + b_\alpha)$) is defined by a single equation of the piecewise-linear function. The piecewise-linear functions $|t - a|$ and $|t - b|$ intersect (or coincide) over this interval at most at one point.

Obviously, the function $F_\alpha(t)$ can be defined by a table consisting of a set of points $\{t_0 - b_\alpha, \ldots, t_{m_{\alpha-1}} - b_\alpha, t_0 + b_\alpha, \ldots, t_{m_{\alpha-1}} + b_\alpha\}$ arranged in nondescending order, and part of points can be eliminated. Consideration is given to the intervals $[t^1, t^2] \in \left[-\sum\limits_{j:=1}^{n} b_j,\ \sum\limits_{j:=1}^{n} b_j\right]$. Consequently, the function $F_\alpha(t)$ will be defined at most by $m_\alpha = 2 \times m_{\alpha-1}$ points. The partition $(B_1(0), B_2(0))$ obtained at the last step $\alpha = n$ is the solution of the problem. The objective function is equal to $F_n(0)$.

2.2. Reduction of the Considered Intervals

Since at the step n one has to calculate the value of the objective function and determine its partition only at the point $t = 0$, at the step $n - 1$ it suffices to calculate the values at the points $t \in [-b_n, b_n]$. Similarly, at the step $n - 2$, it suffices to consider the interval $[-b_n - b_{n-1}, b_n + b_{n-1}]$, and so on. Consequently, it suffices to consider at each step α the interval $\left[-\sum\limits_{j:=\alpha+1}^{n} b_j,\ \sum\limits_{j:=\alpha+1}^{n} b_j\right]$ instead of the interval $\left[-\sum\limits_{j:=1}^{n} b_j,\ \sum\limits_{j:=1}^{n} b_j\right]$.

When constructing the function $F_\alpha(t)$, consideration is given only to the points $t \in \left[-\sum\limits_{j:=\alpha+1}^{n} b_j,\ \sum\limits_{j:=\alpha+1}^{n} b_j\right]$. For the interval to shrink as fast as possible, it is required to arrange b_j in nonascending order. If one takes into account that $F_\alpha(t)$, $\alpha = 1, \ldots, n$, is an even function, then only half of the table suffices for practical realization of the algorithm.

2.3. Example

Let us consider the example of $B = \{100, 70, 50, 20\}$, $n = 4$. The numbers are numerated in nonascending order.

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

Step 1. Arrangement for $b_1 = 100$. Two points $0 + 100$ and $0 - 100$ are considered. The functions are compared over three intervals $[-240, -100]$, $[-100, 100]$, and $[100, 240]$ (or, owing to the reduction of the intervals, over $[-140, -100]$, $[-100, 100]$, and $[100, 140]$), where $240 = \sum\limits_{j=1}^{n} b_j$.

Over the interval $[-240, 0]$ we get the optimal partition $B_1(t) = \{b_1\}$, $B_2(t) = \emptyset$; over the interval $[0, 240]$, the optimal partition $B_1(t) = \emptyset$, $B_2(t) = \{b_1\}$. The results of calculations and the (extended) objective function $F_1(t)$ are shown in Fig. 1. The following information is stored:

-100	100
$(100;)$	$(; 100)$

.

As was already noted, it suffices to store only "half" of the table.

Step 2. Arrangement for $b_2 = 70$. Consideration is given to four points $-100 - 70 = -170$; $-100 + 70 = -30$; $100 - 70 = 30$; and $100 + 70 = 170$. Calculations are carried out over five intervals $[-240, -170]$, $[-170, -30]$, $[-30, 30]$, $[30, 170]$, and $[170, 240]$, but owing to the reduction it suffices to consider only three intervals $[-70, -30]$, $[-30, 30]$, and $[30, 70]$. We get the optimal partition $B_1(t) = \{b_1\}$, $B_2(t) = \{b_2\}$ over the interval $[-70, 0]$ and the optimal partition $B_1(t) = \{b_2\}$, $B_2(t) = \{b_1\}$ over the interval $[0, 70]$. In fact, we do not consider the intervals, but immediately construct the function $F_2(t)$, that is, include in the table the points -30 and 30 and their corresponding partitions. The partitions $B_1(t) = \{b_1\}$, $B_2(t) = \{b_2\}$ and $B_1(t) = \{b_2\}$, $B_2(t) = \{b_1\}$ correspond, respectively, to the points -30 and 30. Figure 2 shows transformation of the function $F_1(t)$ to the functions $F^1(t)$ and $F^2(t)$. The results of calculations and the (extended) objective function $F_2(t)$ are shown in Fig. 3. Therefore, to execute the next algorithmic step, it suffices to store information only at one point $t = 30$:

30
$(70; 100)$

.

Step 3. Arrangement for the number $b_3 = 50$. The four points $-30 - 50 = -80$; $-30 + 50 = 20$; $30 + 50 = 80$; and $30 - 50 = -20$ are considered. Owing to the interval reduction, it suffices

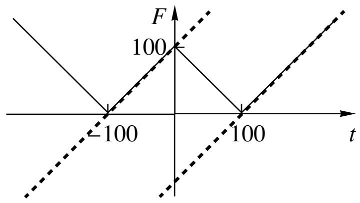

Fig. 1. Function $F_1(t)$.

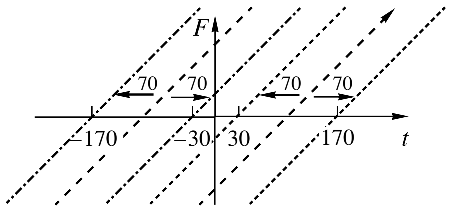

Fig. 2. Transformation of the function $F_1(t)$.

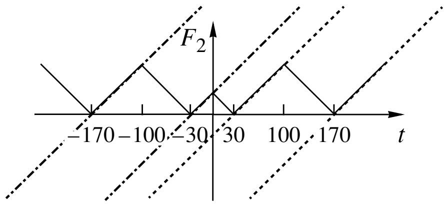

Fig. 3. Function $F_2(t)$.

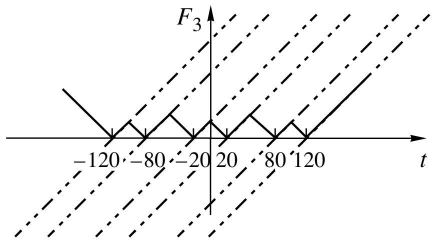

Fig. 4. Function $F_3(t)$.

to consider only one interval $[-20, 20]$. The results of calculations and the (extended) objective function $F_3(t)$ are depicted in Fig. 4. The following information is stored for one point $t = 20$:

20
$(70, 50; 100)$

.

At Step 4 we obtain two optimal "symmetric" solutions $B_1(0) = \{b_1, b_4\}$, $B_2(0) = \{b_2, b_3\}$ and $B_1(0) = \{b_2, b_3\}$, $B_2(0) = \{b_1, b_4\}$.

We have, thus, considered $2(-100\&100)+2(-30\&30)+2(-20\&20)+1(0) = 7$ points, whereas the dynamic programming algorithm [3] (with reduction of intervals) would require $280+140+40+0 = 460$ points. The best of the exact partition algorithms, the *Balsub* algorithm [2, p. 83] requires $O(nb_{\max})$ operations and establishes the solution in $2 \times \frac{n}{2} \times b_{\max} = 400$ operations.

Since $F_\alpha(t)$ is an even function, it suffices to store only "half" of the table. It also deserves noting that with "scaling with small parameter modifications" in the example, that is, with $b'_j = Kb_j + \varepsilon_j$, where $|\varepsilon_j| \ll K$, $j = 1, \ldots, n$, and K is a sufficiently great positive constant, laboriousness of the dynamic programming-based algorithm is $O(Kn \sum b_j)$ operations, whereas that of the graphic algorithm remains the same. For the *Balsub* algorithm with laboriousness of $O(nb_{\max})$ operations, this "scaling" also leads to a K-fold increase in laboriousness. Therefore, the graphic algorithm determines solution by the same number of operations for all points of some cone in the n-dimensional space, provided that the parameters of the example are represented as the point $(b_1, \ldots, b_n)$ in the n-dimensional space. The parameters may be both negative and noninteger.

2.4. Algorithm Laboriousness

Theorem 1. *At the step $\alpha = n$, the graphic algorithm determines the optimal partition $B_1(0)$ and $B_2(0)$.*

Proof. We demonstrate that the algorithm determines the optimal partial partition $B_1(t)$ and $B_2(t)$ of the numerical subset $\{b_1, \ldots, b_\alpha\}$ at each point $t \in \left[-\sum_{j:=\alpha+1}^{n} b_j, \sum_{j:=\alpha+1}^{n} b_j\right]$ of each step $\alpha = 1, \ldots, n$.

The proof is carried out by induction.

(1) Obviously, at the step $\alpha = 1$ we get the optimal partition $B_1(t)$ and $B_2(t)$ at each point $t \in \left[-\sum\limits_{j:=2}^{n} b_j, \sum\limits_{j:=2}^{n} b_j\right]$.

(2) Let us assume that at step α we get some optimal partition $B_1(t)$ and $B_2(t)$ at each point $t \in \left[-\sum\limits_{j:=\alpha+1}^{n} b_j, \sum\limits_{j:=\alpha+1}^{n} b_j\right]$.

(3) We demonstrate that at step $\alpha + 1$ the algorithm provides the optimal partition $B_1(t)$ and $B_2(t)$ at each point $t \in \left[-\sum\limits_{j:=\alpha+2}^{n} b_j, \sum\limits_{j:=\alpha+2}^{n} b_j\right]$.

Ex adverso. Let at the point t the algorithm construct two partitions $(B_1(t - b_{\alpha+1});\ B_2(t - b_{\alpha+1}) \bigcup \{b_{\alpha+1}\})$ and $(B_1(t + b_{\alpha+1}) \bigcup \{b_{\alpha+1}\};\ B_2(t + b_{\alpha+1}))$.

The algorithm takes the partition which provides the least value of $\left|\sum\limits_{b_j \in B_1} b_j + t - \sum\limits_{b_j \in B_2} b_j\right|$.

Let at the point t there be a partition $(\overline{B}_1;\ \overline{B}_2)$ such that

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

and

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

are satisfied. Let $b_{\alpha+1} \in \overline{B}_1$. Then,

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

but the partition $(B_1(t + b_\alpha);\ B_2(t + b_\alpha))$ obtained at the step α at the point $t + b_\alpha$ is not optimal because the partition $(\overline{B}_1 \backslash \{b_{\alpha+1}\}; \overline{B}_2)$ is "better," and we encounter a contradiction. Similar proof is possible for the case of $b_{\alpha+1} \in \overline{B}_2$. $\square$

Analysis of the graphic algorithm suggests the following.

(1) There exists a class of integer examples where the number of points grows exponentially from step to step. For example, $B = \{b_1, \dots, b_n\} = \{M, M-1, M-2, \dots, 1, 1, \dots, 1\}$; M is a sufficiently great number, and the sum of unities in the example is equal to $M(M+1)/2$, that is, $n = M + M(M+1)/2$.

(2) There exists a class of noninteger examples $B = \{b_1, \dots, b_n\}$ where the number of points grows exponentially from step to step. For example, if there exists no set of numbers $\lambda_i = \pm 1$, $i = 1, \dots, n$ such that $\lambda_1 b_1 + \ldots + \lambda_n b_n = 0$ is satisfied, then the number of points in this example grows as $O(2^n)$.

2.5. Experimental Estimation of the Algorithm Laboriousness

The algorithms described in the well-known book [2] were compared with the above graphic algorithm, and two groups of experiments were run.

Table 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

Table 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	10 504	8970
5	16	2191	2490	40	2797	44 675	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	11 056	19	8	82	3849	7039	220	4645	12 112	333	377
9	149	4050	8059	464	5241	13 644	52	21	144	4109	11 803	476	5232	16 200	86	1
10	245	4499	9930	656	5815	16 730	24	10	240	4732	12 410	676	5854	18 840	18	3

(1) In the first group, for all integer values the parameters meet the constraints $40 \geq b_1 \geq \ldots \geq b_n \geq 1$, for $n = 4, 5, \ldots, 10$. The results obtained are compiled in Table 1 where the first column shows problem dimensionality n, the second column, the total number of the examples solved for the given n (the number of combinations of $b_{\max} + n - 1$ numbers n at a time, where $b_{\max} = 40$); the columns three to five show the mean laboriousness, respectively, of the graphic, *Balsab*, and dynamic programming-based algorithms, the columns six to eight show the maximum laboriousness, respectively, of the graphic, *Balsab*, and dynamic programming-based algorithms, the ninth column shows the number of examples for which laboriousness of the *Balsub* algorithm is smaller than that of the graphic algorithm, and the tenth column, the number of examples for which laboriousness of the dynamic programming-based algorithm is smaller than that of the *Balsub* algorithm.

(2) In the second group of experiments, for $n = 4, 5, \ldots, 10$, constructed were groups of 20 000 examples with uniformly selected parameters $b_i \in [1, 200]$, $i = 1, \ldots, n$. Then $1000n$ examples $\{b'_1, \ldots, b'_n\}$ were solved for each example in the n-dimensional space in the neighborhood of $r = 100 + n$ so that $b_i - (100 + n) \leq b'_i \leq b_i + (100 + n)$, $i = 1, \ldots, n$. At that, if a highly laborious example existed in the neighborhood, then "we pass to this example." The process stopped when one fails to find "more complex examples" in the neighborhood. The following results were obtained (see Table 2).

Column one shows the problem dimensionality n, columns two to four show the mean laboriousness of, respectively, the graphic, *Balsub*, and dynamic programming-based algorithms at the "initial point," columns five to seven show the maximum values of the algorithm at the "initial point," columns eight and nine show the maximum and mean numbers of passages from the initial to the final point," columns ten to twelve show the mean laboriousness of the algorithms under consideration at the "final points," columns thirteen to fifteen show the maximal laboriousness of the algorithms under study at the "final points," and columns sixteen and seventeen show the number of examples for which laboriousness of the dynamic programming-based algorithm is smaller than that of the *Balsub* algorithm, respectively, at the initial and final points.

With the exception of 38 examples for $n = 4$ and two examples for $n = 5$ of the 20 000 "final" examples, in all (*Sic!*) "initial" and "final" examples laboriousness of the *Balsub* algorithm exceeded that of the graphic system.

3. GRAPHIC APPROACH TO THE PROBLEM OF ONE-DIMENSIONAL KNAPSACK (0 − 1 *knapsack*)

3.1. Dynamic Programming Algorithm for the Knapsack Problem

The dynamic-programming algorithm based on the Bellman optimality principle [1, 3] is considered as the most efficient one. It works only if the parameters $A, a_i \in Z^+$, $i = 1, \ldots, n$. The function

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

is constructed at each step $\alpha = 1, \ldots, n$ at each point $0 \leq t \leq A$. For each point t, a corresponding $x_\alpha = \arg\max g_\alpha(t)$ is fixed. At the step $\alpha = n$, the optimal solution is determined at the point $t = A$.

The algorithm is illustrated by way of the example from [4, p. 125–129].

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

Its operation is representable as Table 3.

Table 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)

Therefore, we get the optimal solution $(0, 1, 1, 0)$ and the corresponding value of the objective function $g_4(9) = 13$. Laboriousness of the algorithm is $O(nA)$ operations.

3.2. Graphic Approach

The function $g_\alpha(t)$ is representable as

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

that is, for $t \in [t_j, t_{j+1})$ we get $g_\alpha(t) = f_j, j = 0, 1, \ldots, m_\alpha - 1$.

The function $g_{\alpha+1}(t)$ can be obtained from the function $g_\alpha(t)$ in the following way:

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

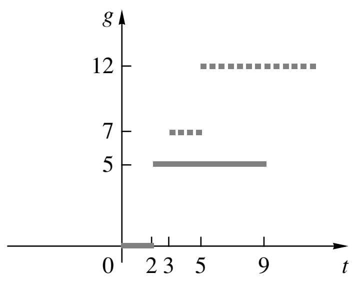

Fig. 5. The functions $g^1(t)$ and $g^2(t)$ (dashed line).

The graph of $g^2(t)$ is constructed from the graph of $g_\alpha(t)$ by an "upward" shift by $c_{\alpha+1}$ and "right" shift by $a_{\alpha+1}$. The function $g^2(t)$ is representable as

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

.

The graph $g^1(t)$ repeats completely that of $g_\alpha(t)$. Consequently, in order to construct $g_{\alpha+1}(t) = \max\{g^1(t), g^2(t)\}$ one has to consider at most $2m_\alpha$ intervals made up by the points from the set $\{t_0, t_1, \ldots, t_{m_\alpha}, t_0 + a_{\alpha+1}, t_1 + a_{\alpha+1}, \ldots, t_{m_\alpha} + a_{\alpha+1}\}$ belonging to the interval $[0, A]$. The number of points does not exceed A for $a_i \in Z^+$, $i = 1, \ldots, n$. For the integer example, laboriousness of the graphic algorithm, therefore, does not exceed $\min\{O(nA), O(nf_{\max})\}$, as it is the case for the dynamic programming-based algorithm. It also deserves noting that the parameters of the problem may be both noninteger and negative. In this case, the graph of $g^2(t)$ is constructed from that of $g_\alpha(t)$ by "downward" shift by $|c_{\alpha+1}|$ (if $c_{\alpha+1} < 0$) and "left" shift by $|a_{\alpha+1}|$ ($a_{\alpha+1} < 0$).

The same example (3) is used below to illustrate the graphic algorithm.

Step 1. As the result, we get the following values:

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

.

Step 2. The dashed lines in Fig. 5 depict the functions $g^1(t)$ and $g^2(t)$. One has to consider the intervals made up by the points $0, 2, 0 + 3, 2 + 3$ in order to construct the function $g_2(t)$.

The results of calculations and the objective function $g_2(t)$ are shown in Fig. 6. As the result:

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

.

Step 3. To construct the function $g_3(t)$, one has to consider the intervals made up by the points $0, 2, 3, 5, 0 + 5, 2 + 5, 3 + 5$. The point $5 + 5 > 9$ is not considered. Many fragments $g^2(t)$ (shown by the dashed line) are "absorbed" and do not participate in $g_3(t)$. The third step of the algorithm results in

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

.

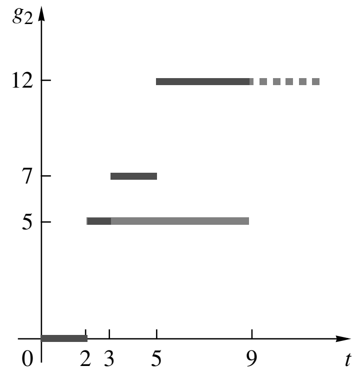

Fig. 6. The objective function $g_2(t)$.

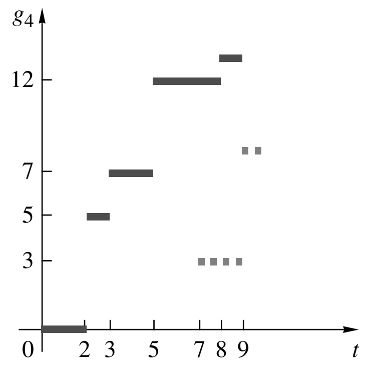

Fig. 7. The objective function $g_4(t)$.

Step 4. The results of calculations and the objective function $g_4(t)$ are depicted in Fig. 7.

One should consider the intervals made up by the points $0, 2, 3, 5, 8, 0+7, 2+7$ in order to construct the function $g_4(t)$. The points $3+7, 5+7, 8+7$ are disregarded. Therefore, it suffices to consider five points. As the result, we obtain

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. Efficiency of the Graphic Algorithm

In the course of its operation the algorithm looked though 17 points: 2 at first step, 3 at the third step, 7 at the third step, and 5 at the fourth step. The dynamic programming algorithm would consider $4 \times 9 = 36$ points. Consequently, for the example at hand we reduce substantially the pseudopolynomial component of algorithm's laboriousness.

It deserves noting that the graphic algorithm works even if $a_i \notin Z$, $\forall i$, $A \notin Z$.

One can easily see that at steps 3 and 4 the number of the "stored" intervals is not doubled. Presumably, laboriousness of the graphic algorithm will be polynomial for a greater number of examples.

To minimize the number of new "stored" intervals appearing at the step α, the source data must be ordered as $\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \ldots \geq \frac{c_n}{a_n}$. Then, the function $g^2(t)$ will be "absorbed" more effectively.

The algorithm takes indirectly into consideration the specificity of the problem. The algorithm of dynamic programming disregards the fact that the least preferable object in the above example is that with the number 4 (see $\frac{c_4}{a_4}$). In the graphic algorithm, one can see at step 4 that the function $g^2(t)$ does not affect $g_4(t)$, that is, some heuristic distinction of the problem was taken into account.

4. CONCLUSIONS

The concept of the graphic approach is a natural continuation of the method of dynamic programming. This approach can be applied to the problems with both noninteger and negative parameters.

It also deserves noting that for $c_i = 2$, $i = 1, \ldots, n$, and $n \geq 4$, the example [5, p. 289]

$$\begin{cases} \sum\limits_{i:=1}^{n} c_i x_i \longrightarrow \max \\ \sum\limits_{i:=1}^{n} 2x_i \leq 2\left[\dfrac{n}{2}\right] + 1, \end{cases} \tag{4}$$

can be solved by the graphic approach in $O(n)$ operations. In the general case, for arbitrary c_i, $i = 1, \ldots, n$, the proposed algorithm solves example (4) in $O(n \log n)$ operations. The algorithms based on the branch-and-bound method need $O(2^{\frac{n}{2}})$ operations to solve this example.

The proposed approach undergoes now an experimental verification with "highly-dimensional" benchmarks. It also will be compared with the algorithms based on the branch-and-bound method (the number of the branching points on the tree compared with the number of the "stored" intervals).

ACKNOWLEDGMENTS

The author would like to thank E.R. Gafarov for valuable discussions of the results obtained.

REFERENCES

1. Papadimitrou, Ch. and Steiglitz, K., *Combinatorial Optimization: Algorithms and Complexity*, Englewood Cliffs: Prentice Hall, 1982. Translated under the title *Kombinatornaya optimizatsiya: algoritmy i slozhnost'*, Moscow: Mir, 1985.
2. Keller, H., Pferschy, U., and Pisinger, D., *Knapsack Problems*, New York: Springer, 2004.
3. Bellman, R., *Dynamic Programming*, Princeton: Princeton Univ. Press, 1957. Translated under the title *Dinamicheskoe programmirovanie*, Moscow: Inostrannaya Literatura, 1960.
4. Sigal, I.Kh. and Ivanova, A.P., *Vvedenie v prikladnoe diskretnoe programmirovanie* (Introduction to Applied Discrete Programming), Moscow: Fizmatlit, 2002.
5. Moiseev, N.N., Ed., *Sovremennoe sostoyanie teorii issledovaniya operatsii* (State-of-the-Art of the Operation Research Theory), Moscow: Nauka, 1979.

This paper was recommended for publication by P.Yu. Chebotarev, a member of the Editorial Board