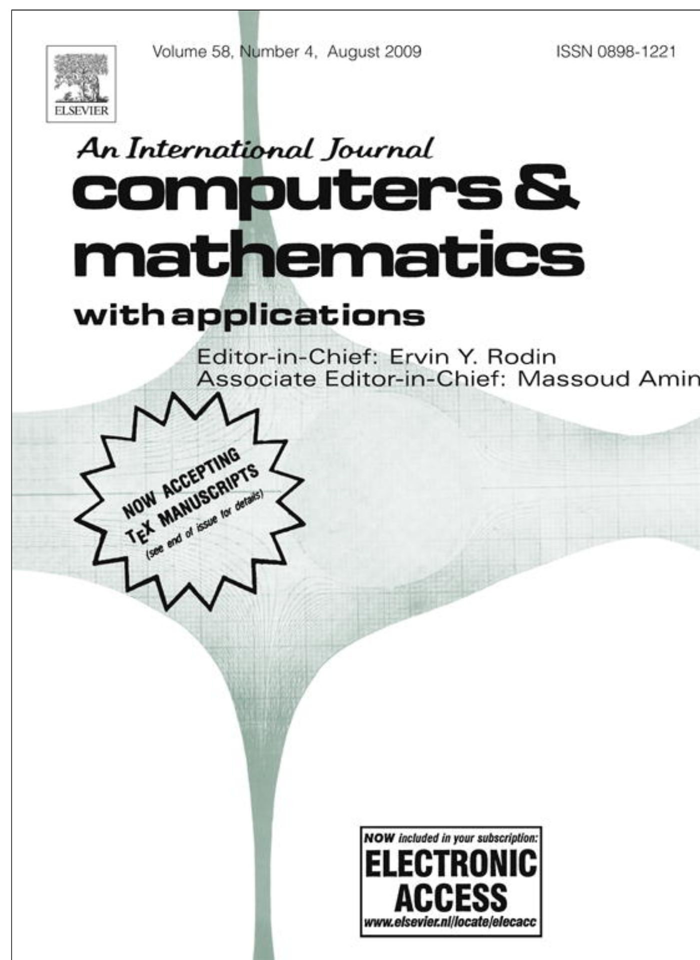
Provided for non-commercial research and education use.
Not for reproduction, distribution or commercial use.



This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/copyright

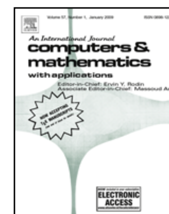
Author's personal copy

Computers and Mathematics with Applications 58 (2009) 619–631

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa



A graphical realization of the dynamic programming method for solving *NP*-hard combinatorial problems

Alexander A. Lazarev [a,*], Frank Werner [b]

[a] Institute of Control Sciences of the Russian Academy of Sciences, Profsoyuznaya street 65, 117997 Moscow, Russia
[b] Fakultät für Mathematik, Otto-von-Guericke-Universität Magdeburg, PSF 4120, 39016 Magdeburg, Germany

ARTICLE INFO

Keywords:
Dynamic programming
Exact algorithm
Graphical algorithm
Partition problem
Knapsack problem

ABSTRACT

In this paper, we consider a graphical realization of dynamic programming. The concept is discussed on the partition and knapsack problems. In contrast to dynamic programming, the new algorithm can also treat problems with non-integer data without necessary transformations of the corresponding problem. We compare the proposed method with existing algorithms for these problems on small-size instances of the partition problem with $n \leq 10$ numbers. For almost all instances, the new algorithm considers on average substantially less "stages" than the dynamic programming algorithm.

Crown Copyright © 2009 Published by Elsevier Ltd. All rights reserved.

1. Introduction

Dynamic programming is a general optimization technique developed by Bellman. It can be considered as a recursive optimization procedure which interprets the optimization problem as a multi-stage decision process. This means that the problem is decomposed into a number of stages. At each stage, a decision has to be made which has an impact on the decision to be made in later stages. By means of Bellman's optimization principle [1], a recursive equation is set up which describes the optimal criterion value at a given stage in terms of the optimal criterion values of the previously considered stage. Bellman's optimality principle can be briefly formulated as follows: Starting from any current stage, an optimal policy for the subsequent stages is independent of the policy adopted in the previous stages. In the case of a combinatorial problem, at some stage α sets of a particular size α are considered. To determine the optimal criterion value for a particular subset of size α, one has to know the optimal values for all necessary subsets of size $\alpha - 1$. If the problem includes n elements, the number of subsets to be considered is equal to $O(2^n)$. Therefore, dynamic programming usually results in an exponential complexity. However, if the problem considered is *NP*-hard in the ordinary sense, it is possible to derive pseudopolynomial algorithms. The application of dynamic programming requires special separability properties.

In this paper, we give a graphical realization of the dynamic programming method which is based on a property originally given for the single machine total tardiness problem by Lazarev and Werner (see [2], Property B-1). This approach can be considered as a generalization of dynamic programming. The new approach often reduces the number of "states" to be considered in each stage. Moreover, in contrast to dynamic programming, it can also treat problems with non-integer data without necessary transformations of the corresponding problem. However, the complexity remains the same as for a problem with integer data in terms of the "states" to be considered.

In the following, we consider the partition and knapsack problems for illustrating the graphical approach. Both these combinatorial optimization problems are *NP*-hard in the ordinary sense (see, e.g. [3–6]). Here, we consider the following formulations of these problems.

Partition problem: Given is an ordered set $B = \{b_1, b_2, \ldots, b_n\}$ of n positive numbers with $b_1 \geq b_2 \geq \cdots \geq b_n$. We wish to determine a partition of the set B into two subsets B^1 and B^2 such that

* Corresponding author.
E-mail addresses: jobmath@mail.ru (A.A. Lazarev), frank.werner@mathematik.uni-magdeburg.de (F. Werner).

0898-1221/$ – see front matter Crown Copyright © 2009 Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2009.06.008

Author's personal copy

$$\left| \sum_{b_i \in B^1} b_i - \sum_{b_i \in B^2} b_i \right| \to \min, \tag{1}$$

where $B^1 \cup B^2 = B$ and $B^1 \cap B^2 = \oslash$.

One-dimensional knapsack problem: One wishes to fill a knapsack of capacity A with items having the largest possible total utility. If any item can be put at most once into the knapsack, we get the binary or $0 - 1$ knapsack problem. This problem can be written as the following integer linear programming problem:

$$\begin{cases} f(x) = \sum_{i=1}^{n} c_i x_i \to \max \\ \sum_{i=1}^{n} a_i x_i \le A; \\ 0 < c_i, 0 < a_i \le A, i = 1, 2, \dots, n; \\ \sum_{i=1}^{n} a_i > A; \\ x_i \in \{0, 1\}, i = 1, 2, \dots, n. \end{cases} \tag{2}$$

Here, c_i gives the utility and a_i the required capacity of item i, $i = 1, 2, \dots, n$. The variable x_i characterizes whether item i is put into the knapsack or not.

We note that problems (1) and (2) are equivalent if

$$c_i = a_i = b_i \quad \text{for } i = 1, 2, \dots, n \quad \text{and} \quad A = \frac{1}{2} \sum_{j=1}^{n} b_j.$$

For the application of the graphical algorithm, the problem data may be arbitrary non-negative real numbers.

This paper is organized as follows. In Section 2, we consider the partition problem. First we explain the concept of the graphical algorithm for this problem. Then we describe in Section 2.2 how the number of intervals (or points) to be considered by the graphical algorithm can be reduced. The algorithm is illustrated by an example in Section 2.3. Then we prove the optimality of the given algorithm and discuss some complexity aspects in Section 2.4. Computational results for small-size instances are given in Section 2.5. In Section 3, we consider the knapsack problem. In Section 3.1, a brief illustration of the application of dynamic programming to the knapsack problem is given. In Section 3.2, the graphical algorithm is applied to the knapsack problem. An illustrative example for the graphical algorithm is given in Section 3.3. Some complexity aspects are discussed in Section 3.4. Finally, we give some concluding remarks in Section 4.

2. Partition problem

2.1. Graphical algorithm

First, we explain the graphical realization of the dynamic programming method for the partition problem. We describe the approach for an arbitrary step (or stage) α: first for $\alpha = 1$ and then for $\alpha = 2, 3, \dots, n$. In each step, we determine function $F_\alpha(t)$ and best partitions $(B^1_\alpha(t); B^2_\alpha(t))$ for the set $\{b_1, b_2, \dots, b_\alpha\}$ in dependence on parameter t by means of the results of the previous step $\alpha - 1$. The value $F_\alpha(t)$ describes the minimal function value (1) for including $b_1, b_2, \dots, b_\alpha$ into one of the current subsets B^1 or B^2 subject to the constraint that in steps $\alpha + 1, \alpha + 2, \dots, n$, altogether t more units of the numbers $b_{\alpha+1}, b_{\alpha+2}, \dots, b_n$ are included into the corresponding set B^1 than into the corresponding set B^2. If t is negative, it means that t more units are included into set B^2 in the next steps.

In the initial step $\alpha = 1$, we have

$$F_1(t) = \begin{cases} -(t + b_1), & \text{if } -\sum_{j=1}^{n} b_j \le t < -b_1; \\ t + b_1, & \text{if } -b_1 \le t < 0; \\ -(t - b_1), & \text{if } 0 \le t < b_1; \\ t - b_1, & \text{if } b_1 \le t \le \sum_{j=1}^{n} b_j \end{cases}$$

and

$$(B^1_1(t); B^2_1(t)) = \begin{cases} (b_1; \oslash), & \text{if } -\sum_{j=1}^{n} b_j \le t < 0; \\ (\oslash; b_1), & \text{if } 0 \le t \le \sum_{j=1}^{n} b_j. \end{cases}$$

Author's personal copy

In order to perform an arbitrary step α, $2 \le \alpha \le n$, we remind that for each t from the interval

$$I_1^n = \left[-\sum_{j=1}^{n} b_j, \sum_{j=1}^{n} b_j \right],$$

the sets $B_{\alpha-1}^1(t)$ and $B_{\alpha-1}^2(t)$ describe a best partition of the first $\alpha - 1$ numbers $b_1, b_2, \ldots, b_{\alpha-1}$ in dependence on parameter t. Moreover, let $\overline{B}_{\alpha-1}^1$ and $\overline{B}_{\alpha-1}^2$ describe an arbitrary but fixed partition of the first $\alpha - 1$ numbers. Thus, in step $\alpha - 1$, we have

$$B_{\alpha-1}^1(t) \cup B_{\alpha-1}^2(t) = \{b_1, b_2, \ldots, b_{\alpha-1}\} = \overline{B}_{\alpha-1}^1 \cup \overline{B}_{\alpha-1}^2 \quad \text{for all } t \in I_1^n.$$

Moreover, from step $\alpha - 1$, we know the function values

$$F_{\alpha-1}(t) = \left| \sum_{b_j \in B_{\alpha-1}^1(t)} b_j + t - \sum_{b_j \in B_{\alpha-1}^2(t)} b_j \right|.$$

Function $F_{\alpha-1}(t)$ is piecewise-linear and, as we see later, it suffices to store those *break points* which are a local minimum of this function:

$$t_1^{\alpha-1}, t_2^{\alpha-1}, \ldots, t_{m_{\alpha-1}}^{\alpha-1}.$$

We give a more detailed discussion later when speaking about the properties of function $F_\alpha(t)$ determined in step α.

In step α, $2 \le \alpha \le n$, the current number b_α is included into one of the sets $\overline{B}_{\alpha-1}^1$ or $\overline{B}_{\alpha-1}^2$. To this end, the algorithm considers the following functions:

$$F_\alpha^1(t) = \left| \sum_{b_j \in B_{\alpha-1}^1(t+b_\alpha)} b_j + t + b_\alpha - \sum_{b_j \in B_{\alpha-1}^2(t+b_\alpha)} b_j \right|,$$

$$F_\alpha^2(t) = \left| \sum_{b_j \in B_{\alpha-1}^1(t-b_\alpha)} b_j + t - b_\alpha - \sum_{b_j \in B_{\alpha-1}^2(t-b_\alpha)} b_j \right|.$$

Then, we construct function

$$\begin{aligned} F_\alpha(t) &= \min \{F_\alpha^1(t), F_\alpha^2(t)\} \\ &= \min\Big\{F_{\alpha-1}(t+b_\alpha), F_{\alpha-1}(t-b_\alpha)\Big\}, \end{aligned}$$

for $t \in I_1^n$. Notice that we have shifted function $F_{\alpha-1}(t)$ to the left and to the right, respectively, by b_α units.

Next, we determine $B_\alpha^1(t)$ and $B_\alpha^2(t)$ according to the best inclusion of number b_α into one of the possible sets. More precisely, for $t \in I_1^n$, if $F_\alpha^1(t) \le F_\alpha^2(t)$, then

$$B_\alpha^1(t) = B_{\alpha-1}^1(t+b_\alpha) \cup \{b_\alpha\} \quad \text{and} \quad B_\alpha^2(t) = B_{\alpha-1}^2(t+b_\alpha)$$

(i.e. b_α is included into the first set), otherwise

$$B_\alpha^1(t) = B_{\alpha-1}^1(t-b_\alpha) \quad \text{and} \quad B_\alpha^2(t) = B_{\alpha-1}^2(t-b_\alpha) \cup \{b_\alpha\}$$

(i.e. b_α is included into the second set). This is analogous to the classical dynamic programming algorithm.

Next, we discuss some properties of function $F_\alpha(t)$. We have already mentioned that function $F_{\alpha-1}(t)$ is known from the previous step by storing the local minimum break points

$$t_1^{\alpha-1}, t_2^{\alpha-1}, \ldots, t_{m_{\alpha-1}}^{\alpha-1}.$$

To describe $F_\alpha(t)$, we use function $F_{\alpha-1}(t-b_\alpha)$ for

$$t \in BP_{\alpha-1} := \{t_1^{\alpha-1}, t_2^{\alpha-1}, \ldots, t_{m_{\alpha-1}}^{\alpha-1}\},$$

i.e. at the points

$$t_1^{\alpha-1} - b_\alpha, \ldots, t_i^{\alpha-1} - b_\alpha, \ldots, t_{m_{\alpha-1}}^{\alpha-1} - b_\alpha,$$

and function $F_{\alpha-1}(t+b_\alpha)$ for $t \in BP_{\alpha-1}$, i.e. at the points

$$t_1^{\alpha-1} + b_\alpha, \ldots, t_i^{\alpha-1} + b_\alpha, \ldots, t_{m_{\alpha-1}}^{\alpha-1} + b_\alpha.$$

Author's personal copy

This gives $m_\alpha \leq 2m_{\alpha-1}$ points, and we will demonstrate that it suffices to consider only these points in step α. We arrange these numbers in non-decreasing order in a set BP_α, i.e. we have

$$BP_\alpha := \{t_1^\alpha, t_2^\alpha, \dots, t_{m_\alpha}^\alpha\}$$

with $t_1^\alpha < t_2^\alpha < \dots < t_{m_\alpha}^\alpha$ (note that possibly a part of points may occur repeatedly but each of such points need to be considered only once so that $m_\alpha < 2m_{\alpha-1}$ is possible).

Considering now the intervals

$$[t_j^\alpha, t_{j+1}^\alpha), \quad j = 1, 2, \dots, m_\alpha - 1,$$

we compare the graphs of both functions $F_{\alpha-1}(t + b_\alpha)$ and $F_{\alpha-1}(t - b_\alpha)$. In each of these intervals, the function $F_{\alpha-1}(t + b_\alpha)$ (and, correspondingly, $F_{\alpha-1}(t - b_\alpha)$) is defined by the same single equation of a piecewise-linear function. Note that two piecewise-linear functions $|t - a|$ and $|t - b|$ intersect (or coincide) in this interval at most at one point. Moreover, the piecewise-linear function $F_\alpha(t)$ obeys the equation $F_\alpha(t) = \left|t - t_i^\alpha\right|$ in the interval

$$\left[t_i^\alpha + \frac{t_{i-1}^\alpha - t_i^\alpha}{2}, t_i^\alpha + \frac{t_{i+1}^\alpha - t_i^\alpha}{2}\right), \quad i = 2, 3, \dots, m_\alpha - 1,$$

i.e. its graph touches the t-axis at the point t_i^α. This means that these break points are local minimum points (in the following denoted as min-break points) and also the zeroes of function $F_\alpha(t)$. As a consequence from the above discussion, function $F_\alpha(t)$ consists of segments having alternately the slopes -1 and $+1$.

Moreover, the same fixed partition $\left(\overline{B}_\alpha^1; \overline{B}_\alpha^2\right)$ is chosen for all t from the whole interval

$$\left[t_i^\alpha + \frac{t_{i-1}^\alpha - t_i^\alpha}{2}, t_i^\alpha + \frac{t_{i+1}^\alpha - t_i^\alpha}{2}\right),$$

that is, we have

$$B_\alpha^k(t') = B_\alpha^k(t'') \quad \text{for all } t', t'' \in \left[t_i^\alpha + \frac{t_{i-1}^\alpha - t_i^\alpha}{2}, t_i^\alpha + \frac{t_{i+1}^\alpha - t_i^\alpha}{2}\right)$$

for $k = 1, 2$ and $i = 2, 3, \dots, m_\alpha - 1$. The local maximum break points of function $F_\alpha(t)$ characterize the values of parameter t, where the chosen partition $(\overline{B}_\alpha^1; \overline{B}_\alpha^2)$ may change (notice that function $F_\alpha(t)$ has alternately local minimum and local maximum break points).

Thus, it suffices to store the min-break points of function $F_\alpha(t)$

$$t_1^\alpha, t_2^\alpha, \dots, t_{m_\alpha}^\alpha,$$

together with the corresponding best partial partitions in a tabular form, i.e. $m_\alpha \leq 2 \cdot m_{\alpha-1}$ points are considered in step α.

Finally, the partition $(B_n^1(0); B_n^2(0))$ obtained in the last step $\alpha = n$ is an optimal solution of the problem. The optimal objective function value is equal to $F_n(0)$.

2.2. Reduction of the considered intervals

Next, we describe how the intervals (and points) to be considered can be reduced. We remind that the numbers in set B are arranged in non-increasing order: $b_1 \geq b_2 \geq \dots \geq b_n$. Moreover, without loss of generality, we assume in the rest of Section 2 that

$$b_1 < \sum_{j=2}^{n} b_j$$

(otherwise, we have a trivial case and $B^1 = \{b_1\}$ and $B^2 = \{b_2, b_3, \dots, b_n\}$ is obviously an optimal solution).

Since in step n, one has to calculate the value of the objective function and to determine the corresponding partition only at point $t = 0$, it suffices to calculate in step $n-1$ the function values $F_{n-1}(t)$ only at the points $t \in I_n^n = [-b_n, b_n]$. Similarly, in step $n - 2$, it suffices to calculate the function values $F_{n-2}(t)$ in the interval $I_{n-1}^n = [-b_n - b_{n-1}, b_n + b_{n-1}]$, and so on. Consequently, it suffices to consider in step α only the interval

$$I_{\alpha+1}^n = \left[-\sum_{j=\alpha+1}^{n} b_j, \sum_{j=\alpha+1}^{n} b_j\right]$$

instead of the interval

$$I_1^n = \left[-\sum_{j=1}^{n} b_j, \sum_{j=1}^{n} b_j\right].$$

If one takes into account that $F_\alpha(t)$, $\alpha = 1, 2, \dots, n$, is an even function, it suffices to store only half of the min-break points and the corresponding best partitions in a table for a practical realization of the algorithm.

Author's personal copy

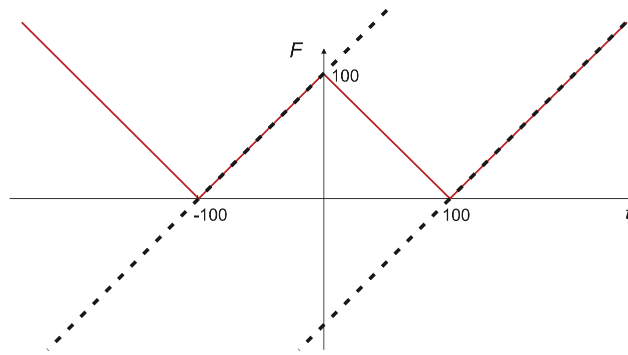


Fig. 1. Function values $F_1(t)$.

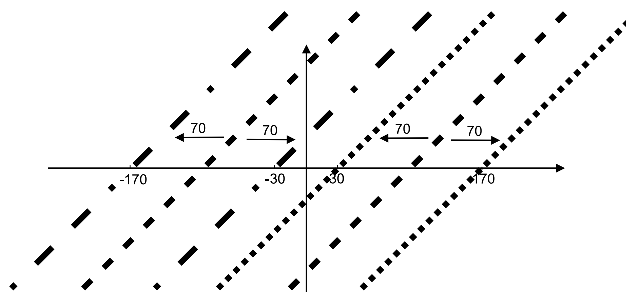


Fig. 2. Transformation of the function $F_1(t)$.

2.3. Example

Let us consider an example with $n = 4$ and $B = \{100, 70, 50, 20\}$. Note that the numbers are given in non-increasing order.

Step 1. We make the arrangement for $b_1 = 100$. We have to consider the two points $0 - 100$ and $0 + 100$. Due to $\sum_{j=1}^{n} b_j = 240$, the function values are compared for the three intervals $[-240, -100)$, $[-100, 100)$, and $[100, 240]$. Taking into account the described reduction of the intervals, it suffices to consider only the intervals $[-140, -100)$, $[-100, 100)$ and $[100, 140]$ due to $\sum_{j=2}^{4} b_j = 140$.

For the interval $[-140, 0)$, we get the optimal partition $B_1^1(t) = \{b_1\}$, $B_1^2(t) = \emptyset$; and for the interval $[0, 140]$, we get the optimal partition $B_1^1(t) = \emptyset$, $B_1^2(t) = \{b_1\}$, i.e. in the max-break point $t = 0$ the optimal partition changes. The results of the calculations and function $F_1(t)$ are shown in Fig. 1. The following information is stored:

t	-100	100
$(B_1^1(t); B_1^2(t))$	$(b_1; \emptyset)$	$(\emptyset; b_1)$

Remember that it suffices to store only "half" of the table and thus, one column can be dropped.

Step 2. We make the arrangement for $b_2 = 70$. We have to consider the four points $-100-70 = -170$, $-100+70 = -30$, $100-70 = 30$ and $100+70 = 170$. Thus, in our calculations we have to take into account the five intervals $[-240, -170)$, $[-170, -30)$, $[-30, 30)$, $[30, 170)$ and $[170, 240]$. Again, due to the reduction of the intervals it suffices to consider only three intervals: $[-70, -30)$, $[-30, 30)$, and $[30, 70]$ due to $\sum_{j=3}^{4} b_j = 70$. Now we get the optimal partition $B_2^1(t) = \{b_1\}$, $B_2^2(t) = \{b_2\}$ for all t from the interval $[-70, 0)$ and the optimal partition $B_2^1(t) = \{b_2\}$, $B_2^2(t) = \{b_1\}$ for all t from the interval $[0, 70]$. In fact, we do not consider the whole intervals, but immediately construct the function $F_2(t)$ by including the points $t = -30$ and $t = 30$ and their corresponding partitions into the table. The partitions $B_2^1(t) = \{b_1\}$, $B_2^2(t) = \{b_2\}$ and $B_2^1(t) = \{b_2\}$, $B_2^2(t) = \{b_1\}$ correspond to the points $t = -30$ and $t = 30$, respectively. Fig. 2 illustrates the transformation of function $F_1(t)$ to the functions $F_2^1(t)$ and $F_2^2(t)$.

The results of the calculations and function $F_2(t)$ are shown in Fig. 3 (for a better illustration, we give function $F_2(t)$ not only for the interval $[-70, 70]$ but for a larger interval including all min-break points). To execute the next step, it suffices to store the information only at the point $t = 30$ (note that point $t = -30$ can be disregarded):

t	30
$(B_2^1(t); B_2^2(t))$	$(b_2; b_1)$

Author's personal copy

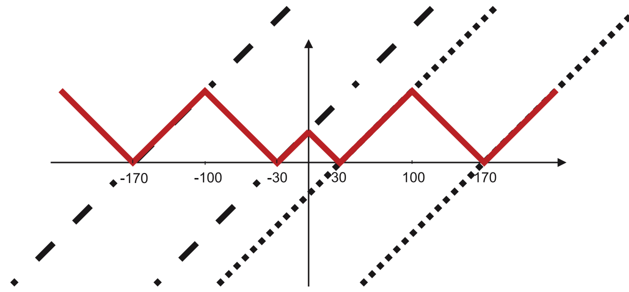


Fig. 3. Function $F_2(t)$.

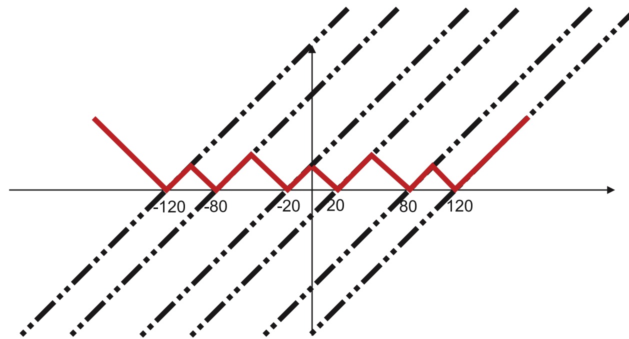


Fig. 4. Function $F_3(t)$.

Step 3. We make the arrangement for the number $b_3 = 50$. We have to consider the four points $-30 - 50 = -80$, $-30 + 50 = 20$, $30 - 50 = -20$ and $30 + 50 = 80$. Due to the possible interval reduction, it suffices to consider only one interval: $[-20, 20]$ due to $\sum_{j=4}^{4} b_j = 20$. The results of the calculations and function $F_3(t)$ are given in Fig. 4. The following information is stored for the point $t = 20$ (point $t = -20$ can be disregarded):

t	20
$(B_3^1(t); B_3^2(t))$	$(b_1; b_2, b_3)$

We have again only one max-break point $t = 0$ in the interval $[-20, 20]$.

Step 4. We obtain the two optimal "symmetric" solutions: $B_4^1(0) = \{b_1, b_4\} = \{100, 20\}$, $B_4^2(0) = \{b_2, b_3\} = \{70, 50\}$ and $B_4^1(0) = \{b_2, b_3\} = \{70, 50\}$, $B_4^2(0) = \{b_1, b_4\} = \{100, 20\}$.

Thus, we have considered $2\,(t = -100$ and $t = 100) + 2(t = -30$ and $t = 30) + 2(t = -20$ and $t = 20) + 1(t = 0) = 7$ points (using that functions $F_\alpha(t)$ are even, we can disregard 3 points whereas the dynamic programming algorithm [1] (with a reduction of intervals) would require $281 + 141 + 41 + 1 = 464$ points. The best known exact partition algorithm *Balsub* [6, p. 83] requires $O(nb_{\max})$ operations and finds a solution with $2 \cdot (n/2) \cdot (b_{\max} + 1) = 404$ operations, where $b_{\max} = \max_{1 \le i \le n} \{b_i\} = 100$.

2.4. Proof of optimality and some complexity aspects

First, we prove that the graphical algorithm finds an optimal solution.

Theorem 1. *In step $\alpha = n$, the graphical algorithm determines an optimal partition $B_n^1(0)$ and $B_n^2(0)$.*

Proof. We show that the algorithm determines an optimal partition $B_\alpha^1(t)$ and $B_\alpha^2(t)$ for the subset $\{b_1, b_2, \dots, b_\alpha\}$ for each point

$$t \in I_{\alpha+1}^n = \left[-\sum_{j=\alpha+1}^{n} b_j, \sum_{j=\alpha+1}^{n} b_j \right]$$

in each step $\alpha = 1, 2, \dots, n$. For $\alpha = n$, we define $I_{n+1}^n := \{0\}$. The proof is done by induction.

(1) Obviously, in step $\alpha = 1$ we get the optimal partition $B_1^1(t)$ and $B_1^2(t)$ for each point

$$t \in I_2^n = \left[-\sum_{j=2}^{n} b_j, \sum_{j=2}^{n} b_j \right].$$

Author's personal copy

(2) Let us assume that in step $\alpha - 1, 2 \le \alpha \le n$, we have found some optimal partition $B^1_{\alpha-1}(t)$ and $B^2_{\alpha-1}(t)$ for each point

$$t \in I^n_\alpha = \left[-\sum_{j=\alpha}^{n} b_j, \sum_{j=\alpha}^{n} b_j \right].$$

(3) We show that in step α, the algorithm also provides an optimal partition $B^1_\alpha(t)$ and $B^2_\alpha(t)$ for each point

$$t \in I^n_{\alpha+1} = \left[-\sum_{j=\alpha+1}^{n} b_j, \sum_{j=\alpha+1}^{n} b_j \right].$$

Let us assume the opposite, i.e. for some point $\bar{t} \in I^n_{\alpha+1}$, the algorithm has constructed two partitions

$$\left(B^1_{\alpha-1}(\bar{t} + b_\alpha) \cup \{b_\alpha\}; B^2_{\alpha-1}(\bar{t} + b_\alpha)\right)$$

and

$$\left(B^1_{\alpha-1}(\bar{t} - b_\alpha); B^2_{\alpha-1}(\bar{t} - b_\alpha) \cup \{b_\alpha\}\right)$$

from which the algorithm selects the partition having the value

$$F_\alpha(\bar{t}) = \min\{F^1_\alpha(\bar{t}), F^2_\alpha(\bar{t})\}.$$

Now assume that this partition is not optimal, i.e. we assume that for this point $\bar{t}$, there exists a partition $(\overline{B}^1_\alpha; \overline{B}^2_\alpha)$ such that

$$\min \left\{ \left| \sum_{b_j \in B^1_{\alpha-1}(\bar{t}+b_\alpha)} b_j + \bar{t} + b_\alpha - \sum_{b_j \in B^2_{\alpha-1}(\bar{t}+b_\alpha)} b_j \right|, \left| \sum_{b_j \in B^1_{\alpha-1}(\bar{t}-b_\alpha)} b_j + \bar{t} - b_\alpha - \sum_{b_j \in B^2_{\alpha-1}(\bar{t}-b_\alpha)} b_j \right| \right\}$$
$$> \left| \sum_{b_j \in \overline{B}^1_\alpha} b_j + \bar{t} - \sum_{b_j \in \overline{B}^2_\alpha} b_j \right|$$

is satisfied. Let $b_\alpha \in \overline{B}^1_\alpha$. Then

$$\left| \sum_{b_j \in B^1_{\alpha-1}(\bar{t}+b_\alpha)} b_j + \bar{t} + b_\alpha - \sum_{b_j \in B^2_{\alpha-1}(\bar{t}+b_\alpha)} b_j \right| > \left| \sum_{b_j \in \overline{B}^1_\alpha \setminus \{b_\alpha\}} b_j + \bar{t} + b_\alpha - \sum_{b_j \in \overline{B}^2_\alpha} b_j \right|,$$

but the partition $(B^1_{\alpha-1}(\bar{t} + b_\alpha); B^2_{\alpha-1}(\bar{t} + b_\alpha))$ obtained in step $\alpha - 1$ for the point $\bar{t} + b_\alpha$ is not optimal since the partition $(\overline{B}^1_\alpha \setminus \{b_\alpha\}; \overline{B}^2_\alpha)$ has a better objective function value which yields a contradiction.

A similar proof can be given for the case $b_\alpha \in \overline{B}^2_\alpha$. □

Next, we give some comments on the complexity of the suggested graphical algorithm.

(1) There exists a class of integer instances, where the number of min-break points grows exponentially. For example, let $B = \{b_1, b_2, \ldots, b_n\} = \{M, M-1, M-2, \ldots, 1, 1, \ldots, 1\}$, where $M > 0$ is a sufficiently large number and there are $M(M+1)/2 + 1$ numbers 1 contained in set B, that is, we have $n = M + M(M+1)/2$. The complexity of the graphical algorithm for this instance is $O(2^M)$.

(2) There exists a class of non-integer instances $B = \{b_1, b_2, \ldots, b_n\}$, where the number of min-break points grows exponentially, too. For example, if there exists no set of numbers $\lambda_i = \pm 1, i = 1, 2, \ldots, n$, such that $\lambda_1 b_1 + \cdots + \lambda_n b_n = 0$ holds, then the number of min-break points in this example grows as $O(2^n)$.

Next, we briefly discuss the situation when the problem data are changed as follows. We consider an instance with $b'_j = Kb_j + \varepsilon_j$, where $|\varepsilon_j| \ll K, j = 1, 2, \ldots, n$, and $K > 0$ is a sufficiently large constant. In this case, the complexity of the dynamic programming algorithm is $O(Kn \sum b_j)$. For the *Balsub* [6] algorithm with a complexity of $O(nb_{\max})$, this “scaling” also leads to an increase in the complexity by factor K. However, the complexity of the graphical algorithm remains the same. In general, the graphical algorithm determines an optimal solution with the same number of operations for all points of some cone in the n-dimensional space, provided that the parameters of the instance are represented as a point $(b_1, b_2, \ldots, b_n)$ in the n-dimensional space.

Author's personal copy

Table 1
Results for the instances of the first group.

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	1677 106 640	115	814	1660	376	1053	3240	83	8 488 253
10	8217 822 536	168	905	2050	500	1172	4000	416	11 426 171

Table 2
Results for the instances of the second group.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
4	8	1463	1591	16	2196	3 080	4	2	6	1310	1 604	20	2207	3 200	10 504	8970
5	16	2191	2490	40	2797	44 675	8	3	17	2482	3 102	40	2811	5 000	6 641	3642
6	29	2700	3586	60	3401	6 570	12	4	26	2884	3 932	60	3418	7 176	3 000	3170
7	50	3145	4881	140	4006	8 729	15	6	54	3353	6 794	136	4029	9 800	1 101	88
8	87	3600	6362	216	4617	11 056	19	8	82	3849	7 039	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

2.5. Computational results

Two algorithms described in [6] (namely the standard dynamic programming algorithm and the *Balsub* algorithm) have been compared with the above graphical algorithm. The complexity of the dynamic programming algorithm is determined by the total number of states to be considered. For the graphical algorithm, the complexity is determined by the number $\sum_{\alpha=1}^{n} m_\alpha$ of min-break points that have to be stored. In the following, we always give the total numbers of min-break points obtained within the interval I_1^n (i.e. without a reduction of the intervals). Notice that, using the reduction of intervals described in Section 2.2, these numbers and thus the complexity will even reduce further. We have run two groups of experiments. Results of the experiments are presented in Tables 1 and 2.

In the first group, we have generated instances for $n = 4, 5, \dots, 10$, where all integer values of the data of the problem satisfy the inequalities $40 \geq b_1 \geq b_2 \geq \cdots \geq b_n \geq 1$. The results obtained are given in Table 1, where the first column shows the value of n, the second column gives the total number of instances solved for the particular value of n (this number is equal to the number of possibilities of choosing n numbers simultaneously from $b_{\max} + n - 1$ numbers, where $b_{\max} = 40$). Columns three to five give the average complexity of the graphical, the *Balsub* and the dynamic programming algorithms, respectively. Columns six to eight present the maximal complexity of the graphical, the *Balsub* and the dynamic programming algorithms, respectively. The ninth column gives the number of instances for which the complexity of the *Balsub* algorithm is smaller than that of the graphical algorithm, and the tenth column gives the number of instances for which the complexity of the dynamic programming algorithm is smaller than that of the *Balsub* algorithm. For all instances, the complexity of the graphical algorithm is essentially smaller than the complexity of the dynamic programming algorithm.

In the second part of experiments, we generated for $n = 4, 5, \dots, 10$ groups of 20 000 instances with uniformly chosen numbers $b_i \in [1, 200], i = 1, 2, \dots, n$. Then $1000 \cdot n$ instances $\{b'_1, b'_2, \dots, b'_n\}$ were solved for each instance in the n-dimensional space in the neighborhood of point $(b_1, b_2, \dots, b_n)$ such that each component can differ by at most $r = 100 + n$ so that

$$b_i - (100 + n) \leq b'_i \leq b_i + (100 + n), \quad i = 1, 2, \dots, n.$$

If an instance with a large number of min-break points (which characterizes the complexity of the graphical algorithm) to be considered occurs in the neighborhood of the current instance data, then we “pass to this instance” and find an instance with a large complexity of the algorithm in the new neighborhood of this instance. The process stops when one fails to find “more difficult instances” in the neighborhood. The results obtained are given in Table 2.

Column one gives the value of n, columns two to four give the average complexity of the graphical, *Balsub*, and the dynamic programming algorithms, respectively, for the “initial instance”. Columns five to seven present the maximal complexity of the algorithm for the “initial instance”. Columns eight and nine present the maximal and average numbers of passages from the “initial to the final instance” in the graphical algorithm. Columns ten to twelve show the average complexity of the algorithms under consideration at the “final instances”. Columns thirteen to fifteen show the maximal complexity of the algorithms considered at the “final instances”. Finally, columns sixteen and seventeen give the numbers of instances for which the complexity of the dynamic programming algorithm is smaller than that of the *Balsub* algorithm at the initial and final instances, respectively.

Among all “initial” and “final” instances, there were only 38 instances with $n = 4$ and two instances with $n = 5$ among the “final” instances, where the graphical algorithm had to consider more states (i.e. min-break points) than the *Balsub* algorithm.

Author's personal copy

Table 3
Application of the dynamic programming algorithm.

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)

3. Knapsack problem

Recall that in the binary knapsack problem we are given a knapsack with capacity A and n items $i = 1, 2, \ldots, n$ with utility values c_i and weights a_i:

$$\begin{cases} f(x) = \sum_{i=1}^{n} c_i x_i \to \max \\ \sum_{i=1}^{n} a_i x_i \le A; \\ 0 < c_i,\ 0 < a_i \le A,\ i = 1, 2, \ldots, n; \\ \sum_{i=1}^{n} a_i > A; \\ x_i \in \{0, 1\}, \quad i = 1, 2, \ldots, n. \end{cases}$$

The variable x_i characterizes whether item i is put into the knapsack or not.

3.1. Illustration of dynamic programming

The dynamic programming algorithm based on Bellman's optimality principle [1,5] is considered to be the most efficient one. It is assumed that all parameters are integer: A; $a_i \in Z^+$, $i = 1, 2, \ldots, n$. In step α, $\alpha = 1, 2, \ldots, n$, the function values

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

are calculated for each integer point (i.e. "state") $0 \le t \le A$. Here, we have $g_0(t) = 0$ for all integers t with $0 \le t \le A$. For each point t, a corresponding best (partial) solution $(x_1(t), x_2(t), \ldots, x_\alpha(t))$ is stored.

The algorithm is illustrated by the example from [7, p. 125–129]:

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

The results with the dynamic programming algorithm are summarized in Table 3. Therefore, for $t = 9$, we get the optimal solution

$$x(13) = (x_1(13), x_2(13), x_3(13), x_4(13)) = (0, 1, 1, 0)$$

and the corresponding optimal objective function value $g_4(9) = 13$. The complexity of the algorithm is $O(nA)$.

3.2. Graphical approach

We describe the modifications of the graphical algorithm. In an arbitrary step α, $2 \le \alpha \le n$, function $g_\alpha(t)$ is now defined for all real t with $0 \le t \le A$. In particular, $g_\alpha(t)$ is a step function, i.e. it is a discontinuous function with jumps. We assume that the function values $g_{\alpha-1}(t_j) = f_j$, $j = 1, 2, \ldots, m_{\alpha-1}$ are known from the previous step $\alpha - 1$. Here, t_1 is the left boundary point to be considered and $t_2, t_3, \ldots, t_{m_{\alpha-1}}$ are the *jump points* of function $g_{\alpha-1}(t)$ from the interval $[0, A]$. These function values can be stored in a tabular form as follows:

t	t_1	t_2	$\ldots$	$t_{m_{\alpha-1}}$
$g(t)$	f_1	f_2	$\ldots$	$f_{m_{\alpha-1}}$

Author's personal copy

As a consequence, for $t \in [t_j, t_{j+1})$, we have $g_{\alpha-1}(t) = f_j$ for all $j = 1, 2, \dots, m_{\alpha-1} - 1$. Note that $t_1 = 0$ and $f_1 = 0$ for all $\alpha = 1, 2, \dots, n$. In the initial step $\alpha = 1$, we get

$$g_1(t) = \begin{cases} 0, & \text{if } 0 \le t < a_1, \\ c_1, & \text{if } a_1 \le t \le A. \end{cases}$$

Similarly to dynamic programming, function $g_\alpha(t)$ can be obtained from function $g_{\alpha-1}(t)$ in the following way:

$$g^1(t) = g_{\alpha-1}(t);$$

$$g^2(t) = \begin{cases} g^1(t), & \text{if } 0 \le t < a_\alpha, \\ c_\alpha + g_{\alpha-1}(t - a_\alpha), & \text{if } a_\alpha \le t \le A. \end{cases}$$

Then

$$g_\alpha(t) = \max\{g^1(t), g^2(t)\}$$

and for the corresponding solution component, we get

$$x_\alpha(t) = \begin{cases} 1, & \text{if } g^2(t) > g^1(t), \\ 0, & \text{otherwise.} \end{cases}$$

In the last step ($\alpha = n$), we have an optimal solution of the problem for each point $t \in R$ with $0 \le t \le A$:

$$x(t) = (x_1(t), x_2(t), \dots, x_n(t)).$$

Consider an arbitrary step α with $2 \le \alpha \le n$, The graph of $g^2(t)$ can be constructed from the graph of $g_{\alpha-1}(t)$ by an "upward" shift by c_α and a "right" shift by a_α. Function $g^2(t)$ can be stored in a tabular form as follows:

t	$t_1 + a_\alpha$	$t_2 + a_\alpha$	...	$t_{m_{\alpha-1}} + a_\alpha$
$g(t)$	$f_1 + c_\alpha$	$f_2 + c_\alpha$	...	$f_{m_{\alpha-1}} + c_\alpha$

The graph of function $g^1(t)$ corresponds to that of $g_{\alpha-1}(t)$. Consequently, in order to construct

$$g_\alpha(t) = \max\{g^1(t), g^2(t)\},$$

one has to consider at most $2m_{\alpha-1}$ points (intervals) obtained by means of the points chosen from the set

$$\{t_1, t_2, \dots, t_{m_{\alpha-1}}, t_1 + a_\alpha, t_2 + a_\alpha, \dots, t_{m_{\alpha-1}} + a_\alpha\}$$

belonging to the interval $[0, A]$. The number of points does not exceed A for $a_j \in Z^+$, $j = 1, 2, \dots, n$. For the integer example, the number of intervals considered by the graphical algorithm in step α, therefore, does not exceed $\min\{O(nA), O(nf_{\max})\}$, where $f_{\max} = \max_{1 \le i \le m_\alpha}\{f_i\}$. Thus, the worst case complexity is the same as for the dynamic programming algorithm.

3.3. Example

We use again instance (3) to illustrate the graphical algorithm by an example.

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

t	0	2
$g_1(t)$	0	5
$x(t)$	(0,,,)	(1,,,)

Step 2. Due to $a_2 = 3$, one has to consider the intervals obtained from the boundary points $0, 2, 0 + 3, 2 + 3$ in order to construct the function $g_2(t)$. The dashed lines in Fig. 5 give function $g^2(t)$. As the result, we get:

t	0	2	3	5
$g_2(t)$	0	5	7	12
$x(t)$	(0, 0,,)	(1, 0,,)	(0, 1,,)	(1, 1,,)

Step 3. To construct function $g_3(t)$, one has to consider the intervals obtained from the boundary points $0, 2, 3, 5, 0+5, 2+5, 3+5$ due to $a_3 = 5$. The point $5 + 5 > 9$ need not to be considered. The results of the calculations and function $g_3(t)$ are shown in Fig. 6. Several fragments $g^2(t)$ (shown by the bright line) are "absorbed" and do not influence the values $g_3(t)$. In the third step of the algorithm, we obtain the following results:

t	0	2	3	5	8
$g_3(t)$	0	5	7	12	13
$x(t)$	(0, 0, 0,)	(1, 0, 0,)	(0, 1, 0,)	(1, 1, 0,)	(0, 1, 1,)

Step 4. The results of the calculations and the objective function $g_4(t)$ are depicted in Fig. 7. One has to consider the intervals resulting from the boundary points $0, 2, 3, 5, 8, 0+7, 2+7$ in order to construct function $g_4(t)$. The points $3+7, 5+7, 8+7$

Author's personal copy

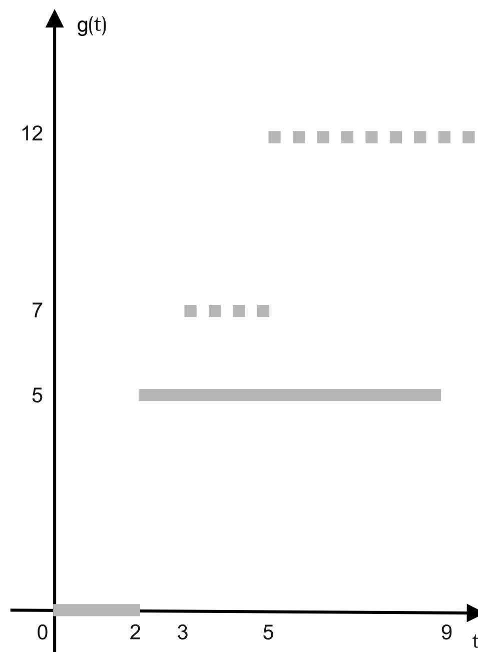


Fig. 5. Functions $g^1(t)$ and $g^2(t)$ (dashed lines).

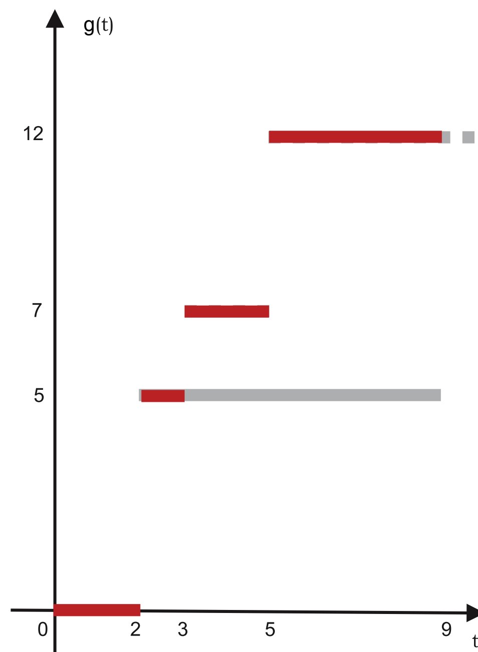


Fig. 6. Function $g_3(t)$.

need not to be considered since they are larger than $A = 9$. Therefore, it suffices to consider five points. As the result, we obtain the following table:

t	0	2	3	5	8
$g_4(t)$	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)

3.4. Some complexity aspects of the graphical algorithm

The complexity of the graphical algorithm for solving the knapsack problem is determined by the total number of jump points $\sum_{\alpha=1}^{n} m_\alpha - n$ to be considered. Notice that there are $m_\alpha - 1$ jump points and the left boundary point in each step α.

Author's personal copy

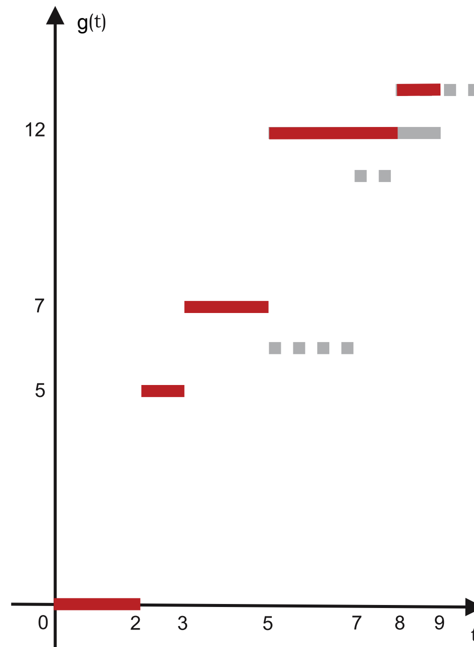


Fig. 7. Function $g_4(t)$.

For the example presented in Section 3.3, the graphical algorithm stored 12 jump points: 1 in the first step ($t = 2$), 3 in the second step ($t = 2, 3, 5$), 4 in the third step ($t = 2, 3, 5, 8$), and 4 in the last step ($t = 2, 3, 5, 8$), too. In contrast, the dynamic programming algorithm would consider $4 \times 9 = 36$ points. Consequently, for the example at hand the number of required steps is substantially reduced.

One can see that for the example considered in Section 3.3, in steps 3 and 4 the number of intervals to be considered is not doubled. One can conjecture that the average number of necessary operations of the graphical algorithm (i.e. total number of jump points) will be polynomial for a substantial number of instances.

To minimize the number of new intervals to be considered in each step α, the source data must be ordered such that

$$\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \cdots \geq \frac{c_n}{a_n}.$$

In this case, the functions $g^2(t)$ will be "absorbed" more effectively.

Moreover, the graphical algorithm takes the specific properties of the problem indirectly into account. The dynamic programming algorithm disregards the fact that the least preferable item in the above example is that with number 4 (see the quotient c_4/a_4). In the graphical algorithm, one can see that function $g^2(t)$ does not affect $g_4(t)$ in step 4. Thus, heuristic considerations can be taken into account and may reduce the complexity of the algorithm.

Moreover, we also note that for $c_i = 2$, $i = 1, 2, \ldots, n$, and $n \geq 4$, the example from [4,8]

$$\begin{cases} \sum_{i=1}^{n} c_i x_i \rightarrow \max \\ \sum_{i=1}^{n} 2x_i \leq 2\left[\frac{n}{2}\right] + 1 \end{cases} \tag{4}$$

can be solved by the graphical approach in $O(n)$ operations. In the general case, i.e. for arbitrary c_i, $i = 1, 2, \ldots, n$, the proposed algorithm solves example (4) with $O(n \log n)$ operations. An algorithm based on the branch and bound method needs $O(\frac{2^n}{\sqrt{n}})$ operations to solve this example, when $c_i = 2$ for all $i = 1, \ldots, n$.

4. Conclusions

The concept of the graphical approach is a natural generalization of the dynamic programming method. This algorithm can also treat instances with non-integer data without increasing the number of required operations (in terms of min-break or jump points to be considered). For small-size problems, it turned out to be superior to the standard algorithms in terms of the average and maximum complexity, i.e. the average and maximal number of "states" (break points for the partition problem and jump points for the knapsack problem) to be considered.

Note that the complexity of the graphical algorithm is the same for the following instances of the partition problem with $n = 3 : \{1, 3, 2\}; \{1, 100, 99\}$ and $\{10^{-6}, 1, 1 - 10^{-6}\}$. In contrast, the complexity of the dynamic programming algorithm

Author's personal copy

strongly differs for the above small instances. In particular, the third instance requires a scaling of the data since integer "states" are considered. However, this considerably increases the complexity.

For further research, it will be interesting to compare the graphical algorithm with the existing dynamic programming-based algorithms on benchmark problems with larger size. In addition, a comparison with branch and bound algorithms is of interest, too. More precisely, one can compare the number of nodes of the branching tree with the number of points to be considered by the graphical algorithm.

Acknowledgments

This work has been supported by DAAD (Deutscher Akademischer Austauschdienst A/08/08679, Ref. 325) and the program of Presidium of Russian Academy of Sciences N 29 "The mathematical theory of control". The authors would like to thank Dr. E.R. Gafarov for valuable discussions of the results obtained.

References

[1] R. Bellman, Dynamic Programming, Princeton Univ. Press, Princeton, 1957.
[2] A. Lazarev, F. Werner, Algorithms for special cases of the single machine total tardiness problem and an application to the even-odd partition problem, Mathematical and Computer Modelling 49 (9–10) (2009) 2061–2072.
[3] A.A. Korbut, J.J. Finkelstein, Discrete Programming, Nauka, Moscow, 1969, (in Russian) (In German: Korbut A.A., Finkelstein, J.J., Diskrete Optimierung, Berlin: Akademie-Verlag, 1971).
[4] J.J. Finkelstein, Approximative Methods and Applied Problems of Discrete Programming, Nauka, Moscow, 1976, (in Russian).
[5] Ch. Papadimitrou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice Hall, Englewood Cliffs, 1982.
[6] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, Springer, New York, 2004.
[7] I.Kh. Sigal, A.P. Ivanova, Introduction to Applied Discrete Programming, Fizmatlit, Moscow, 2007, (in Russian).
[8] N.N. Moiseev (Ed.), State-of-the-Art of the Operations Research Theory, Nauka, Moscow, 1979, (in Russian).