European Journal of Operational Research 270 (2018) 435–447

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

Approximation schemes for non-separable non-linear boolean programming problems under nested knapsack constraints

Nir Halman[a,1], Hans Kellerer[b], Vitaly A. Strusevich[c,*]

[a] The Hebrew University of Jerusalem, Israel
[b] Institut für Statistik und Operations Research, Universität Graz, Universitätsstraße 15, Graz A-8010, Austria
[c] Department of Mathematical Sciences, University of Greenwich, Old Royal Naval College, Park Row, Greenwich, London SE10 9LS, UK

ARTICLE INFO

Article history:
Received 18 May 2017
Accepted 7 April 2018
Available online 12 April 2018

Keywords:
Combinatorial optimization
Non-linear boolean programming
Geometric rounding
K-approximation sets and functions
FPTAS

ABSTRACT

We consider a fairly general model of "take-or-leave" decision-making. Given a number of items of a particular weight, the decision-maker either takes (accepts) an item or leaves (rejects) it. We design fully polynomial-time approximation schemes (FPTASs) for optimization of a non-separable non-linear function which depends on which items are taken and which are left. The weights of the taken items are subject to nested constraints. There is a noticeable lack of approximation results on integer programming problems with non-separable functions. Most of the known positive results address special forms of quadratic functions, and in order to obtain the corresponding approximation algorithms and schemes considerable technical difficulties have to be overcome. We demonstrate how for the problem under consideration and its modifications FPTASs can be designed by using (i) the geometric rounding techniques, and (ii) methods of *K*-approximation sets and functions. While the latter approach leads to a faster scheme, the running times of both algorithms compare favorably with known analogues for less general problems.

1. Introduction

One of the most popular types of decision in business decision-making is related to accepting or rejecting a certain activity. In this paper, informally we call this types of decisions "take-or-leave" decisions. This, for example, happens in make-or-buy situations, when a particular product could be either manufactured internally or bought from outside. More meaningful examples are contained below.

We address Boolean programming problems, in which it is required to either minimize or maximize a non-linear function that facilitates the leave-or-take decision making. Suppose we are given a set $N = \{1, 2, \ldots, n\}$ of items, so that each item j is associated with a positive integer α_j, which we call its weight. For a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ consisting of n Boolean components, such a leave-or-take function can be written as

$$Z(\mathbf{x}) = \sum_{j=1}^{n} f_j\left(\sum_{i=1}^{j} \alpha_i x_i\right) x_j + \sum_{j=1}^{n} g_j\left(\sum_{i=1}^{j} \alpha_i (1 - x_i)\right)(1 - x_j). \quad (1)$$

Here we assume that $x_j = 1$ if item j is taken, and $x_j = 0$ if item j is left. The functions f_j represent the cost or penalty for taking items; they are non-decreasing non-negative functions that depend on total accumulated weight of the taken items i, $1 \le i \le j$. On the other hand, the functions g_j represent the cost or penalty for leaving items; they are non-decreasing non-negative functions that depend on total accumulated weight of all left items i, $1 \le i \le j$. Addressing the issues of approximability of function $Z(\mathbf{x})$ further in this paper we make assumptions on computability of the functions f_j and g_j, $1 \le j \le n$. Besides, further in this section we present several examples of problems from various application areas which reduce to minimizing function $Z(\mathbf{x})$ of the form (1) subject to additional linear constraints.

We start with the examples that involve a knapsack constraint

$$\sum_{j=1}^{n} \alpha_j x_j \le A. \quad (2)$$

Notice that the presented examples are given here for illustration only; in fact the results contained in the paper concern optimization of (1) under more general constraints than (2).

* Corresponding author.
E-mail addresses: halman@huji.ac.il (N. Halman), hans.kellerer@uni-graz.at (H. Kellerer), V.Strusevich@greenwich.ac.uk (V.A. Strusevich).

[1] The first author was partially supported by Israel Science Foundation, Grant 399/17.

https://doi.org/10.1016/j.ejor.2018.04.013

1.1. Safe helicopter pickup

In the offshore petroleum industry, employees are transported by helicopters to and from offshore installations. Assume that set N is a set of installations, and α_j people who have to be picked up from an installation j. We have a helicopter $H1$ of total capacity A and another helicopter $H2$ of sufficient capacity. In a safe helicopter transportation model, the risk of visiting an installation j is measured by a function that depends on the type of aircraft used and on the number of people on board the aircraft that take off at installation j and land at the next installation of the route; see Qian, Strusevich, Gribkovskaia, and Halskau (2015) and Rustogi and Strusevich (2013) for detailed descriptions of the model and approaches to its solution for simple risk-measuring functions. If installation j is visited by helicopter $H1$, then $x_j = 1$; otherwise, $x_j = 0$. For helicopter $H1$, the risk is measured by a function f_j, and for helicopter $H2$ function g_j is used. In either case, the argument of each of these functions is the total number of people that take off at installation j. It is required to decide which installation is visited by which aircraft so as to minimize total risk.

1.2. Two-chamber holding

Assume that set N is a set of orders, so that they arrive one order per time period and order j consists of α_j items to be put on hold. The holding facility consists of two chambers, one of capacity A and the other of sufficient capacity. The functions f_j and g_j measure the holding costs in the respective chamber, that depends on the total number of items currently on hold in that chamber. The purpose is to decide to which chamber to place an order so that total holding cost is minimized.

1.3. Production with dirt accumulation

The quality of equipment often deteriorates as it is used due to accumulation of unwanted by-products. This is, for example, observed when a floor sanding machine operates and saw dust is accumulated. Formally, assume that set N is a set of jobs to be processed on a single machine. Processing job j accumulates α_j units of dirt. No more than A units of dirt can be accumulated; after that cleaning is required. The cleaning operation takes constant time. The actual processing time of job j is defined either by function f_j if the job is processed in the first group, before the cleaning, or by function g_j, if the job is processed in the second group, after the cleaning. In either case, the function depends on the amount of dirt generated by all previously scheduled jobs of the group. The difference in these functions can be explained by the fact that a cleaning operation does not necessarily return the machine to the initial "as good as new" state. The purpose is to split the jobs into two groups so as to minimize the makespan, i.e., the maximum completion time of all jobs.

We next elaborate on the hardness of minimizing function (1) under knapsack constraints and the need for its approximation. Clearly, the problem of minimizing function (1) subject to a knapsack constraint is no easier that the famous linear knapsack problem and is therefore at least NP-hard in the ordinary sense; see Kellerer, Pferschy, and Pisinger (2004). This is why in this paper we study a possibility of developing approximation schemes for the problems of optimizing (1) subject to linear constraints. Consider the function of n Boolean variables

$$S(\mathbf{x}) = \sum_{1\le i<j\le n} \alpha_i\beta_j x_i x_j + \sum_{1\le i<j\le n} \alpha_i\beta_j (1-x_i)(1-x_j) + \sum_{j=1}^{n} \mu_j x_j + \sum_{j=1}^{n} \nu_j (1-x_j) + \Gamma, \tag{3}$$

which has been a popular object of study. The function is called symmetric quadratic function, because both the quadratic and the linear parts of the objective function are separated into two terms, one depending on the variables x_j, and the other depending on the variables $(1-x_j)$. Following Kellerer and Strusevich (2010a,b), we call the problem of minimizing the objective (3) subject to the linear knapsack constraint (2) the Symmetric Quadratic Knapsack Problem. That problem is known to be an underlying mathematical model for many scheduling problems; see the focused surveys Kellerer and Strusevich (2012, 2016). Notice that the non-separable quadratic terms in (3) are special cases of the corresponding non-linear terms in (1).

In turn, the symmetric quadratic function is a variant of the well-studied non-separable quadratic function known as the half-product. The latter function has been introduced by Badics and Boros (1998) and can be written as

$$H(\mathbf{x}) = \sum_{1\le i<j\le n} \alpha_i\beta_j x_i x_j - \sum_{j=1}^{n} \gamma_j x_j. \tag{4}$$

There are numerous publications on the design and analysis of problems of optimizing function (4) and its variants, with and without an additive constant, as well as with and without linear constraints; see surveys Kellerer and Strusevich (2012, 2016). The key issue of these studies has been design and analysis of fully polynomial-time approximation schemes.

For a collection of decision variables $\mathbf{x}$, consider a problem of minimizing a function $\varphi(\mathbf{x})$ that takes positive values. Recall that a polynomial-time algorithm that finds a feasible solution $\mathbf{x}^H$ such that $\varphi(\mathbf{x}^H)$ is at most $\rho \ge 1$ times the optimal value $\varphi(\mathbf{x}^*)$ is called a *ρ−approximation* algorithm; the value of ρ is called a *worst-case ratio bound*. A family of ρ−approximation algorithms is called a *fully polynomial-time approximation scheme (FPTAS)* if $\rho = 1+\varepsilon$ for any $\varepsilon > 0$ and the running time is polynomial with respect to both the length of the problem input and $1/\varepsilon$. If a function $\varphi(\mathbf{x})$ takes both positive and negative values, then an FPTAS delivers a feasible solution $\mathbf{x}^H$ such that $\varphi(\mathbf{x}^H) - \varphi(\mathbf{x}^*) \le \varepsilon|\varphi(\mathbf{x}^*)|$. The latter definition is applicable to the problem of minimizing the half-product function (4).

The main problem studied in this paper can be formulated as follows.

$$\text{Minimize} \quad Z = \sum_{j=1}^{n} f_j\left(\sum_{i=1}^{j} \alpha_i x_i\right) x_j + \sum_{j=1}^{n} g_j\left(\sum_{i=1}^{j} \alpha_i (1-x_i)\right)(1-x_j)$$

$$\text{Subject to} \quad \sum_{j=1}^{k} \alpha_j x_j \le d_k, \quad 1 \le k \le n,$$

$$x_j \in \{0,1\}, \quad j = 1,2,\ldots,n. \tag{5}$$

The set of constraints for this problem is not just a single linear knapsack constraint of the form (2) but a set of n nested linear constraints with the right-hand sides forming a non-decreasing sequence $d_1 \le d_2 \le \cdots \le d_n$. As above, here for each j, $1 \le j \le n$, functions f_j and g_j are non-negative non-decreasing functions of a positive argument.

Nested restrictions with a triangle matrix of constraints are often use in mathematical programming. For instance, in non-linear resource allocation problems with submodular constraints there is class of problems called (Nested) which is represented by the triangle matrix; see Hochbaum and Hong (1995) .

Below, we illustrate the relevance of the problem (5) by linking it to several problems.

1.4. Single-item lot-sizing problem

The single-item lot-sizing problem is among the most popular problems of combinatorial optimization; see the recent survey

by Brahimi, Absi, Dauzère-Pérès, and Nordli (2017). The classical lot-sizing problem involves minimizing the sum of the production costs and the holding costs of items of a single product over a given number of periods to satisfy the demand. Typically, there are two sets of decision variables that for each period represent the number of the produced items and the number of the held items. These variables are in general non-negative integers. Among the results obtained for the general lot-sizing problem are fully polynomial approximation schemes, see e.g., Chubanov, Kovalyov, and Pesch (2006, 2008) and Halman, Orlin, and Simchi-Levi (2012, end of Section 6.1).

Since in our model we are concerned with the Boolean variables, below we follow Hardin, Nemhauser, and Savelsbergh (2007) and describe a model with $0-1$ decision variables. Given n periods, define a variable y_i such that $y_i = 1$ if no production takes place in period i, $1 \le i \le n$; otherwise $y_i = 0$ and that means that during period i exactly c_i items are produced. Let d_i denote the demand for period i, $1 \le i \le n$. Introduce

$$C_j = \sum_{i=1}^{j} c_i, \;\; D_j = \sum_{i=1}^{j} d_i, \;\; 1 \le j \le n,$$

where C_j is the number of items that could be produced in periods $1, 2, \ldots, j$ and D_j is the aggregated demand in these periods. To satisfy the demand, the inequalities

$$\sum_{i=1}^{j} c_i(1 - y_i) \ge D_j$$

must hold for all j, $1 \le j \le n$. They can be rewritten as nested constraints

$$\sum_{i=1}^{j} c_i y_i \le C_j - D_j, \;\; 1 \le j \le n.$$

Let f_j be the cost function of producing c_j items in period j, while h_j be the cost function of all held items, i.e., those items that have been manufactured by period j on top of the demand D_j, $1 \le j \le n$. Notice that in the model studied in Hardin et al. (2007) the production cost function is linear and no holding cost is taken into consideration.

The resulting problem can be formulated as

$$\begin{aligned} \text{Minimize} \;\; & Z = \sum_{j=1}^{n} f_j\big(c_j(1 - y_j)\big) + \sum_{j=1}^{n} h_j\left(C_j - D_j - \sum_{i=1}^{j} c_i y_i\right) \\ \text{Subject to} \;\; & \sum_{i=1}^{j} c_i y_i \le C_j - D_j, \;\; 1 \le j \le n, \\ & y_i \in \{0, 1\}, \;\; i = 1, 2, \ldots, n. \end{aligned} \tag{6}$$

1.5. Single machine scheduling with rejection

Assume that set N is a set of jobs to be processed on a single machine, owned by the decision-maker. Each job $j \in N$ has the processing time p_j and the deadline d_j by which it must be completed. The jobs are supposed to be numbered in non-decreasing order of their deadlines. The decision-maker may either accept a job to process internally or reject a job, e.g., by subcontracting it. In the former case, define the decision variable $x_j = 1$; otherwise, $x_j = 0$. The cost of processing of each accepted job j is defined by $f_j(C_j)$, where $C_j = \sum_{i=1}^{j} p_j x_j$ is its completion time. The other jobs are given to be processed to the subcontractor, and the cost of handling the rejected job is defined by function g_j. In the simplest case, g_j can be just a positive number β_j that represents the rejection penalty of job j; see, e.g., Shabtay, Gaspar, and Kaspi (2013) and Kellerer and Strusevich (2013) for examples of scheduling problems with simple rejection penalties. In a more general case g_j is a non-decreasing function that depends on the processing of all rejected jobs i, $1 \le i \le j \le n$.

$$\begin{aligned} \text{Minimize} \;\; & Z = \sum_{j=1}^{n} f_j\left(\sum_{i=1}^{j} p_i x_i\right) x_j + \sum_{j=1}^{n} g_j\left(\sum_{i=1}^{j} p_i(1 - x_i)\right)(1 - x_j) \\ \text{Subject to} \;\; & \sum_{j=1}^{k} p_j x_j \le d_k, \;\; 1 \le k \le n, \\ & x_j \in \{0, 1\}, \;\; j = 1, 2, \ldots, n. \end{aligned} \tag{7}$$

It can be seen that problems (6) and (7) share many features with the problem (5), the main problem of this study.

The main outcome of this paper is that we demonstrate that under some reasonable additional conditions on functions f_j and g_j, which, e.g., hold when the functions are polynomials of a fixed degree, the problem (5) admits an FPTAS.

The remainder of this paper is organized as follows. In Section 2, we briefly review general principles and known approaches to the design of FPTASs for the problems of integer nonlinear programming. In Section 3, we show that problem (5) is solvable by a pseudopolynomial-time dynamic programming algorithm and such an algorithm can be easily converted into an FPTAS by using a popular geometric rounding technique. An alternative approach to designing an FPTAS for problem (5) is presented in Section 4. Here we show that the powerful technique of K-approximation sets and functions developed by Halman, Klabjan, Li, Orlin, and Simchi-Levi (2014) can be adapted to converting another dynamic programming algorithm to an FPTAS. While the second FPTAS is faster, the running time of each scheme, although not strongly polynomial, is still computationally acceptable, and the proofs of their correctness are relatively simple. In Section 5, we discuss various extensions to the basic model, i.e., handling more general cost functions and nested knapsack constraints. Moreover, we show that very similar principles can be applied to developing an FPTAS for the maximization counterpart of problem (5). Section 6 contains concluding remarks.

2. Approaches to FPTAS design

In this section, we review most influential results on designing approximation schemes for solving problems of non-linear Boolean programming.

Since the pioneering works by Ibarra and Kim (1975), Sahni (1977) and Lawler (1979), the development of fully polynomial-time approximation schemes for various combinatorial optimization problems has become a major direction of research. From the point of view of accuracy of the found solution, an FPTAS is the best approximation result one may expect for an NP-hard problem. An FPTAS provides a piece of evidence that the problem under consideration allows finding a solution arbitrarily close to the optimum. Still, many researchers consider an FPTAS to be an algorithm of a limited practical value due to its fairly large time and space requirements. On the other hand, for many problems, e.g., various versions of the linear knapsack problems, there are approximation schemes that show a good computational behavior for instances of practical interest. Examples include an FPTAS for the linear knapsack problem with the running time of $O(n \log(\frac{1}{\varepsilon}) + \frac{(\log(1/\varepsilon))^2}{\varepsilon^3})$ by Kellerer and Pferschy (1999, 2004) and an FPTAS for the subset-sum problem with the running time of $O(n + \frac{\log(1/\varepsilon)}{\varepsilon^2})$ and the space requirement of $O(n + \frac{1}{\varepsilon})$ by Kellerer, Mansini, Pferschy, and Speranza (2003). References to the papers that report positive results on computational experiments with FPTASs are contained in the survey Kovalyov and Kubiak (2012).

Virtually all known FPTASs are obtained by converting a dynamic programming (DP) algorithm available for solving the

problem under consideration. Typically, such a DP algorithm requires a pseudopolynomial running time and using various algorithmic techniques (e.g., trimming of the state space, rounding of the input data or the output values, etc.) the time is appropriately reduced, while the accuracy is (insignificantly) lost.

Multiple attempts have been made to identify general principles and techniques for developing an FPTAS. Below we briefly and informally review several papers which we see as the most influential ones regarding the design of FPTASs, and at the same time most relevant to our study.

Woeginger (2000) provides a series of conditions on a DP algorithm such that the algorithm converts into a FPTAS. A problem that admits such a DP algorithm is called *DP-benevolent*, and each DP-benevolent problem is proved to admit an FPTAS. The paper describes several subclasses of the DP-benevolent problems for which an FPTAS can be developed within a simpler framework than in the general case. Multiple examples of DP-benevolent problems are given and many previously known results are shown to be consequences of the DP-benevolence of the corresponding problems. On the other hand, Woeginger (2000) demonstrates that several versions of the linear knapsack problem do not exhibit the DP-benevolence and in fact do not admit an FPTAS unless $P = NP$.

Among examples given in Woeginger (2000) is the problem of minimizing the completion time variance on a single machine. The problem is not known to exhibit the DP-benevolence, and although it admits an FPTAS, the scheme is developed based on different principles. It is interesting to point out that the problem of minimizing the completion time variance can be reformulated in terms of minimizing the half-product function (4); see Badics and Boros (1998). Thus, the problem of minimizing the half-product is not known to be DP-benevolent, although it admits an FPTAS with a running time of $O(n^2/\varepsilon)$ given by Erel and Ghosh (2008). The same holds for various versions of the latter problem, for example, for minimizing a so-called positive half-product function

$$P(\mathbf{x}) = \sum_{1\le i<j\le n} \alpha_i\beta_j x_i x_j + \sum_{j=1}^{n} \mu_j x_j + \sum_{j=1}^{n} \nu_j\left(1-x_j\right) + \Gamma, \tag{8}$$

with positive coefficients introduced by Janiak, Kovalyov, Kubiak, and Werner (2005). The best known FPTAS for minimizing a convex positive half-product is due to Kellerer and Strusevich (2013). See Kellerer and Strusevich (2012, 2016) for reviews of similar results. We note that it is not clear whether minimizing (1) under knapsack constraints exhibits DP-benevolence.

Kovalyov and Kubiak (2012) study the class PT_{opt} of problems for which an objective function is defined over partitions of a finite set of items into a given number of subsets. They formulate four quite natural and fairly easily verifiable conditions which guarantee that if these conditions hold for a problem of class PT_{opt}, then such a problem admits an FPTAS.

Among problems that admit an FPTAS due to the conditions established in Kovalyov and Kubiak (2012) are two problems related to minimizing a version of the half-product objective (4) subject to a linear knapsack constraint. One of these functions is a positive half-product function (8), and the other is a non-separable quadratic function similar to (3) with the coefficients in the two quadratic terms being not the same. For both functions, all coefficients are positive and an appropriate sum U of these coefficients serves as an upper bound. The FPTAS obtained in Kovalyov and Kubiak (2012) for minimizing function (8) require $O(\log^3(U)\log(\max\{\log U,\ n,\ 1/\varepsilon\}n^3/\varepsilon^2))$ time, while that for minimizing a generalization of function (3) takes $O(\log^4(U)\log(\max\{\log U,\ n,\ 1/\varepsilon\}n^5/\varepsilon^4))$ time.

The results presented in Kovalyov and Kubiak (2012) are quite relevant to this paper, since function (1) can be seen as defined over partitions of a set of n items into two subsets (taken and left items). However, even if the objective (1) satisfies all required conditions, it is unlikely that the resulting FPTAS will have the running time faster than $O(\log^4(U)\log(\max\{\log U,\ n,\ 1/\varepsilon\}n^5/\varepsilon^4))$, even if the nested constraints are simplified to a single knapsack constraint.

Notice that the problem of minimizing a convex function of the form (8) under the knapsack constraint (2) admits an FPTAS that requires $O(n^2/\varepsilon)$ time; see Kellerer and Strusevich (2013). Besides, the problem of minimizing the symmetric quadratic function (3) under the same constraint (2) admits an FPTAS that can be implemented in $O(n^4\max\{\log n,\ 1/\varepsilon^2\})$ time, as shown by Xu (2012) who extends the technique developed in Kellerer and Strusevich (2010a,b). Notice that the running times of these FPTASs are strongly polynomial and they are developed by totally different approaches than those outlined by Woeginger (2000) and Kovalyov and Kubiak (2012).

For minimizing a (quasi-)concave function of Boolean variables under linear constraints, several authors explore the fact that a (quasi-)concave function of continuous variables achieves its minimum at an extremal point of a polytope. If a (quasi-)concave objective function is of a low rank k, i.e., depends on k input vectors, then for its minimization over a polytope several FPTASs are available; see Goyal and Ravi (2013) and Mittal and Schulz (2013). A special case of such a function of rank 2 is the product of two linear functions, which has multiple applications in combinatorial optimization; see Kern and Woeginger (2007) and Goyal, Genc-Kaya, and Ravi (2011) for FPTASs for minimization of such an objective. The running time of all mentioned approximation schemes is not strongly polynomial.

Halman et al. (2014) develop a powerful framework for converting DP algorithms for FPTASs for a wide range of problems of deterministic and stochastic optimization, provided that the objective function is separable. Their approach is based on establishing two novel sets of computational rules, which the authors call the calculus of K -approximation functions and K-approximation sets. This approach is illustrated on ten problems, for almost all of them no FPTAS has been previously known. Among the problems which are especially relevant to this study is a generalized non-linear knapsack problem

$$\begin{aligned} &\text{Maximize} \quad \sum_{j=1}^{n} \pi_j\left(x_j\right) \\ &\text{Subject to} \quad \sum_{j=1}^{n} \nu_j\left(x_j\right) \le B, \\ &\qquad\qquad l_j \le x_j \le u_j,\ x_j \in \mathbb{Z}^+,\ j = 1,2,\ldots,n. \end{aligned} \tag{9}$$

to maximize a separable non-linear objective function subject to a non-linear knapsack constraint. The decision variables are non-negative integers that have individual upper and lower bounds. Problem (9) has been known to admit an FPTAS, provided either the functions π_j are concave and the functions ν_j are convex (see Hochbaum, 1995) or these functions are monotone (see Kovalyov, 1996). It is shown in Halman et al. (2014) that problem (9) admits an FPTAS provided that functions π_j and ν_j are non-decreasing and the length of any of their values under the binary encoding is polynomially bounded by the length of the problem's input. They also consider the minimization counterpart of this problem. These two problems are among the most general problems of integer programming with a single constraint that are known to admit an FPTAS, provided that the objective function is separable.

While the framework of Halman et al. (2014) cannot be applied "as is" to problem (5) due to the non-separability of the objective function, we show in Section 4 how to apply their technique of K-approximation sets and functions in order to design an FPTAS for it. That section also contains a brief review of the general framework based on the technique of K-approximation sets and

functions. Moreover, the approach can further be adapted to handling more general problems than problem (5).

In the forthcoming sections, we show that the DP algorithms available for problem (5) and its maximization counterpart can be converted into FPTASs. This is done in a surprisingly simple way, and the analysis of the performance of these schemes is rather elementary. Moreover, the running time of our FPTASs, although not strongly polynomial, can be seen as computationally acceptable.

3. Minimization problem: FPTAS by geometric rounding

In this section, we describe a version of a dynamic programming (DP) algorithm for solving problem (5). Then we show that under certain conditions the DP algorithm can be converted into an approximation scheme that behaves as an FPTAS.

In the DP algorithms below, the decision variables x_j are scanned in the order of their numbering and are given either the value of 1 (an item is taken) or 0 (an item is left).

Define

$$A_k = \sum_{j=1}^{k} \alpha_j, \; k = 1, 2, \ldots, n. \tag{10}$$

and suppose that the values $x_1, x_2, \ldots, x_k$ have been assigned. One version of our DP algorithm deals with partial solutions associated with states of the form

$$(k, Z_k, y_k),$$

where

k is the number of the assigned variables;
Z_k is the current value of the objective function;
$y_k := \sum_{j=1}^{k} \alpha_j x_j$ is the state variable, whose value is the total weight of the taken items.

Let us call the states of the form (k, Z_k, y_k), i.e., states whose state variable is y_k, the *primal* states, and let the DP algorithm that manipulates the primal states be called the *primal* algorithm. The primal DP algorithm is used further in this paper as a basis for designing an FPTAS for the maximization counterpart of problem (5). Its formal description is given in Section 5.4.

For obtaining an FPTAS for problem (5), in which the objective has to be minimized, it is convenient to use another form of the DP algorithm that manipulates the states of the dual form $(k, Z_k, \hat{y}_k)$, where k and Z_k have the same meaning as above, while the state variable is $\hat{y}_k = A_k - y_k$. It is clear that $\hat{y}_k$ is the total weight of the considered items that have not been taken.

We refer to the DP algorithm for solving problem (5) that manipulates dual states of the form $(k, Z_k, \hat{y}_k)$ as *dual* algorithm. Its formal statement is given below.

Algorithm DDP

Step 1. Start with the initial state $(0, Z_0, \hat{y}_0) = (0, 0, 0)$. Compute the values $A_k, k = 1, 2, \ldots, n$, by (10).

Step 2. For all k from 0 to $n-1$ do

Make transitions from each stored state of the form

$$\left(k, Z_k, \hat{y}_k\right), \tag{11}$$

into the states of the form

$$\left(k+1, Z_{k+1}, \hat{y}_{k+1}\right) \tag{12}$$

by assigning the next variable x_{k+1}.

(a) Define $x_{k+1} = 1$, provided that it is feasible to take item $k+1$, i.e., if the $(k+1)$-th nested constraint $A_{k+1} - \hat{y}_k \le d_{k+1}$ holds. If feasible, the assignment $x_{k+1} = 1$ changes a state of the form (11) to a state of the form (12), where

$$\hat{y}_{k+1} = \hat{y}_k; \;\; Z_{k+1} = Z_k + f_{k+1}\left(A_{k+1} - \hat{y}_{k+1}\right), \tag{13}$$

(b) Define $x_{k+1} = 0$, which is always feasible. This assignment changes a state of the form (11) into the state of the form (12) such that

$$\hat{y}_{k+1} = \hat{y}_k + \alpha_{k+1}; \;\; Z_{k+1} = Z_k + g_{k+1}\left(\hat{y}_{k+1}\right). \tag{14}$$

Step 3. Output the optimal value of the function that corresponds to the smallest value of Z_n among all found states of the form $(n, Z_n, \hat{y}_n)$.

To develop an FPTAS for problem (5) we need certain assumptions regarding computability and properties of functions f_j and g_j:

- each function f_j and g_j, $1 \le j \le n$, can be computed in constant time;
- there exists a constant $r \ge 1$ such that for each j, $1 \le j \le n$, and for any positive u

$$g_j(ux) \le u^r g_j(x). \tag{15}$$

Notice that these assumptions hold, e.g., if the functions f_j and g_j are polynomials of a fixed degree that does not exceed r in the case of the functions g_j. Notice that if (15) holds as the equality then the corresponding function is homogeneous. Moreover, inequality (15) with $r = 1$ represents the relation known as the decreasing returns to scale.

The role of the assumption (15) is discussed further in this section, after the presentation of the FPTAS and the proof of its correctness. Since the argument of functions g_j is the total weight of the non-taken items, it is natural to base the FPTAS on conversion of Algorithm DDP which uses variables $\hat{y}_k$, $1 \le k \le n$, as state variables.

Additionally, throughout the paper we assume that for a given positive ε a power of $1+\varepsilon$ can be computed in constant time.

In the description and the analysis of the FPTAS the following upper bound

$$Z^{UB} = \sum_{j=1}^{n} f_j\left(d_j\right) + \sum_{j=1}^{n} g_j\left(A_j\right) \tag{16}$$

is used. The algorithm below splits the range of $\hat{y}$-values and the range of Z-values into subintervals with the endpoints that form geometric sequences.

Algorithm EpsMin1

Step 1. Compute Z^{UB} by (16) and A_k, $1 \le k \le n$, by (10). For a given positive ε, introduce the intervals, whose endpoints form geometric sequences. For the $\hat{y}-$ values, introduce the intervals

$$[0,0], \left[1, (1+\varepsilon)^{\frac{1}{rn}}\right], \left[(1+\varepsilon)^{\frac{1}{rn}}, (1+\varepsilon)^{\frac{2}{rn}}\right], \ldots, \left[(1+\varepsilon)^{\frac{u-1}{rn}}, A_n\right]$$

where u is the largest integer such that $\left\lceil (1+\varepsilon)^{\frac{u-1}{rn}} \right\rceil \le A_n$. Call these intervals I_ℓ, $\ell = 0, 1, \ldots, u$. For the $Z-$values, introduce the intervals

$$[0,0], \left[1, (1+\varepsilon)^{\frac{1}{n}}\right], \left[(1+\varepsilon)^{\frac{1}{n}}, (1+\varepsilon)^{\frac{2}{n}}\right],$$
$$\left[(1+\varepsilon)^{\frac{2}{n}}, (1+\varepsilon)^{\frac{3}{n}}\right], \ldots \left[(1+\varepsilon)^{\frac{v-1}{n}}, Z^{UB}\right],$$

where v is the largest integer such that $\left\lceil (1+\varepsilon)^{\frac{v-1}{n}} \right\rceil \le Z^{UB}$. Call these intervals J_t, $t = 0, 1, \ldots, v$.

Step 2. Store the initial state (0,0,0). For each k, $0 \le k \le n-1$, do the following:

According to Algorithm DDP move from a stored dual state $(k, Z_k, \hat{y}_k)$ to at most two feasible dual states of the form $(k+1, Z_{k+1}, \hat{y}_{k+1})$, where $Z_{k+1} \le Z^{UB}$, using the relations (13) and (14). If the number of generated states $(k+$

$1, Z_{k+1}, \hat{y}_{k+1}$) with the Z-values in the same interval J_t and with the $\hat{y}$-values in the same interval I_ℓ exceeds one, then keep only one of these states, that with the largest $\hat{y}$-value.

Step 3. Among all values Z_n found in Step 2 identify the smallest one. Starting from a state associated with this value of Z_n, perform backtracking to find the corresponding decision variables x_j, $j = 1, \ldots, n$. Compute the value of the objective function with the found x_j's, call this value Z^ε and accept it as an approximate value of the objective function.

We now analyze the performance of Algorithm EpsMin1.

Lemma 1. *Assume that the dynamic programming Algorithm DDP is applied to problem (5) that satisfies (15) and finds a chain of dual states*

$$(0, 0, 0), (1, Z_1^*, \hat{y}_1^*), \ldots, (n, Z_n^*, \hat{y}_n^*)$$

leading to the optimal value $Z^* = Z_n^*$. Then for each k, $1 \le k \le n$, Algorithm EpsMin1 finds a state $\left(k, Z_k, \hat{y}_k\right)$ such that

$$\hat{y}_k^* \le \hat{y}_k \le (1+\varepsilon)^{\frac{k}{m}} \hat{y}_k^* \tag{17}$$

and

$$Z_k \le (1+\varepsilon)^{\frac{k}{n}} Z_k^*. \tag{18}$$

Proof. The proof is by induction. To establish the basis of induction for $k = 1$, notice the following. If $x_1^* = 1$ then take $\hat{y}_1 = 0$, $Z_1 = Z_1^* = f_1(\alpha_1)$, while if $x_1^* = 0$ then take $\hat{y}_1 = \hat{y}_1^* = \alpha_1, Z_1 = g_1(\alpha_1)$. The conditions (17) and (18) hold for $k = 1$.

Assume that the lemma holds for all k, $1 \le k \le q < n$. In the optimal chain of dual states, in accordance with (13) and (14) we have that for $k = q$ a state $(q+1, Z_{q+1}^*, \hat{y}_{q+1}^*)$ is computed, where

$$\hat{y}_{q+1}^* = \hat{y}_q^* + \alpha_{q+1}\left(1 - x_{q+1}^*\right).$$

This state is obviously feasible, i.e., $A_{q+1} - \hat{y}_{q+1}^* \le d_{q+1}$.

Take a stored state $(q, Z_q, \hat{y}_q)$ and consider a state $(q+1, \tilde{Z}_{q+1}, \tilde{y}_{q+1})$ obtained from it by the transformation

$$\tilde{y}_{q+1} = \hat{y}_q + \alpha_{q+1}\left(1 - x_{q+1}^*\right).$$

It follows from (17) applied with $k = q$ that

$$\tilde{y}_{q+1} \ge \hat{y}_q^* + \alpha_{q+1}\left(1 - x_{q+1}^*\right) = \hat{y}_{q+1}^*, \tag{19}$$

i.e., $A_{q+1} - \tilde{y}_{q+1} \le A_{q+1} - \hat{y}_{q+1}^* \le d_{q+1}$. This implies that state $(q+1, \tilde{Z}_{q+1}, \tilde{y}_{q+1})$ is feasible and will be contained among states computed in Step 2.

We also deduce from (17) applied with $k = q$ that

$$\begin{aligned}\tilde{y}_{q+1} &\le (1+\varepsilon)^{\frac{q}{m}} \hat{y}_q^* + \alpha_{q+1}\left(1 - x_{q+1}^*\right) \\ &\le (1+\varepsilon)^{\frac{q}{m}} \left(\hat{y}_q^* + \alpha_{q+1}\left(1 - x_{q+1}^*\right)\right) = (1+\varepsilon)^{\frac{q}{m}} \hat{y}_{q+1}^*.\end{aligned} \tag{20}$$

If $x_{q+1}^* = 1$, then it follows from (19) and (18) for $k = q$ as well as from the monotonicity of function f_{q+1} that

$$\begin{aligned}\tilde{Z}_{q+1} &= Z_q + f_{q+1}\left(A_{q+1} - \tilde{y}_{q+1}\right) \le (1+\varepsilon)^{\frac{q}{n}} Z_q^* + f_{q+1}\left(A_{q+1} - \hat{y}_{q+1}^*\right) \\ &\le (1+\varepsilon)^{\frac{q}{n}} \left(Z_q^* + f_{q+1}\left(A_{q+1} - \hat{y}_{q+1}^*\right)\right) = (1+\varepsilon)^{\frac{q}{n}} Z_{q+1}^*.\end{aligned} \tag{21}$$

If $x_{q+1}^* = 0$, then it follows from (20) and (18) for $k = q$ as well as from the monotonicity and the main property (15) of function g_{q+1} that

$$\begin{aligned}\tilde{Z}_{q+1} &= Z_q + g_{q+1}\left(\tilde{y}_{q+1}\right) \le (1+\varepsilon)^{\frac{q}{n}} Z_q^* + g_{q+1}\left((1+\varepsilon)^{\frac{q}{m}} \hat{y}_{q+1}^*\right) \\ &\le (1+\varepsilon)^{\frac{q}{n}} Z_q^* + (1+\varepsilon)^{\frac{q}{n}} g_{q+1}\left(\hat{y}_{q+1}^*\right) = (1+\varepsilon)^{\frac{q}{n}} Z_{q+1}^*.\end{aligned} \tag{22}$$

Thus, in any case,

$$\tilde{Z}_{q+1} \le (1+\varepsilon)^{\frac{q}{n}} Z_{q+1}^*. \tag{23}$$

If state $\left(q+1, \tilde{Z}_{q+1}, \tilde{y}_{q+1}\right)$ is kept as a state $\left(q+1, Z_{q+1}, \hat{y}_{q+1}\right)$, i.e., if we define $\hat{y}_{q+1} := \tilde{y}_{q+1}$ and $Z_{q+1} = \tilde{Z}_{q+1}$, then (17) and (18) hold for $k = q+1$.

If state $\left(q+1, \tilde{Z}_{q+1}, \tilde{y}_{q+1}, \right)$ is not kept, then there exists a feasible state $\left(q, Z_{q+1}, \hat{y}_{q+1}\right)$ such that both values $\tilde{Z}_{q+1}$ and Z_{q+1} belong to the same interval J_t, while both values $\hat{y}_{q+1}$ and $\tilde{y}_{q+1}$ belong to the same interval I_ℓ and $\hat{y}_{q+1} > \tilde{y}_{q+1}$. Since $\hat{y}_{q+1}^* \le \tilde{y}_{q+1}$, we have that $\hat{y}_{q+1}^* \le \hat{y}_{q+1}$ as required by (17). Besides, it follows that if two $\hat{y}$-values belong to the same interval I_ℓ, then their ratio never exceeds the ratio of its endpoints equal to $(1+\varepsilon)^{\frac{1}{m}}$, and we derive from (20) that

$$\hat{y}_{q+1} \le (1+\varepsilon)^{\frac{1}{m}} \tilde{y}_{q+1} \le (1+\varepsilon)^{\frac{q+1}{m}} \hat{y}_{q+1}^*.$$

Similarly, the ratio between the values $\tilde{Z}_{q+1}$ and Z_{q+1} does not exceed the length of the interval J_t, so that due to (23) we have that

$$Z_{q+1} \le (1+\varepsilon)^{\frac{1}{n}} \tilde{Z}_{q+1} \le (1+\varepsilon)^{\frac{q+1}{n}} Z_{q+1}^*.$$

Hence, (18) holds for $k = q+1$. □

Theorem 1. *For problem (5) that satisfies (15), Algorithm EpsMin1 is an FPTAS that requires $O\left(\frac{n3}{\varepsilon^2} \log A_n \log Z^{UB}\right)$ time.*

Proof. By Lemma 1 Algorithm EpsMin1 outputs a state $(n, Z_n, \hat{y}_n)$, and due to property (18) for $k = n$ we have that

$$Z_n \le (1+\varepsilon)^{\frac{n}{n}} Z_n^* = (1+\varepsilon) Z_n^*.$$

Thus, the algorithm delivers the required quality of approximation. Let us estimate its running time. Computing Z^{UB} in Step 1 takes $O(n)$ time. Since r is a constant, the numbers of used intervals can be estimated as $u = O\left(n \log_{1+\varepsilon} A_n\right)$ and $v = O\left(n \log_{1+\varepsilon} Z^{UB}\right)$. In each of n iterations the number of kept states does not exceed uv, and at most $2uv$ new states are created of which at most uv are kept. The overall running time is $O(n + nuv) = O\left(n^3 \log_{1+\varepsilon} A_n \log_{1+\varepsilon} Z^{UB}\right)$. Since for any positive b the equality $\log_{1+\varepsilon} b = O\left(\frac{1}{\varepsilon} \log b\right)$ holds, we obtain the running time of $O\left(\frac{n^3}{\varepsilon^2} \log A_n \log Z^{UB}\right)$, which is polynomial (but not strongly polynomial) with respect to the length of the problem's input. □

There are several reasons why the assumption (15) turns out to be essential. First, the constant r helps us to define the intervals for the $\hat{y}$-variables and to make sure that their number, u, is appropriately bounded; see the proof of Theorem 1. Second, one of the crucial points in the proof of Lemma 1 is to demonstrate that $\tilde{Z}_{q+1} \le (1+\varepsilon)^{\frac{q}{n}} Z_{q+1}^*$. Since $\tilde{y}_{q+1} \ge \hat{y}_{q+1}^*$, to derive the chain of inequalities (21) we need no assumptions on the behavior of function f_j, except its monotonicity. On the other hand, to derive the chain of inequalities (22), we need to rely on (15).

Notice that the running time stated in Theorem 1 holds, provided that computing each function f_j and g_j takes constant time. The algorithm still behaves as an FPTAS if we assume that such computation requires polylogarithmic time. The latter assumption is widely used in the analysis of the FPTASs for problems of nonlinear optimization, see, e.g., Halman et al. (2014).

4. Minimization problem: FPTAS by *K*-approximation sets and functions

In this section, we give another FPTAS for problem (5), which is faster with respect to each of the terms n and ε, and is based on the technique of finding K-approximation sets and functions briefly discussed in Section 2. We first formulate a DP algorithm, then review the K-approximation sets and functions technique, and finally design and analyze an FPTAS.

4.1. A DP formulation

Let y, $0 \le y \le d_n$, denote a possible value of the total weight of all taken items; we refer to this value as the available *space*. The improved DP algorithm presented below manipulates states of the form

$$(k, z_k(y), a_k(y), b_k(y)),$$

where for $0 \le k \le n$ and $0 \le y \le d_n$, the state variables are as follows:

- k, is the number of considered items, for which the "take-or-leave" decisions have been made;
- $z_k(y)$, $0 \le k \le n$, is the optimal value of the objective (5), provided that only items $1, \ldots, k$ have been considered and the available space is $\min\{y, d_k\}$;
- $a_k(y) \le y$ is the *actual* used space, i.e., the total weight of the taken items in the optimal solution associated with $z_k(y)$;
- $b_k(y) \le A_k$ is the total weight of the left items in the optimal solution associated with $z_k(y)$; as above, A_k is defined by (10).

The improved DP algorithm can be stated as follows.

Algorithm DPab

Step 1. Start with the initial states $(0, z_0(y), a_0(y), b_0(y))$, $0 \le y \le d_n$, defined by

$$z_0(y) = a_0(y) = b_0(y) = 0, \quad y = 0, \ldots, d_n.$$

Step 2. For all k from 1 to n find the states $(k, z_k(y), a_k(y), b_k(y))$, as follows:

(a) Compute

$$z'(y) = f_k(a_{k-1}(y) + \alpha_k) + z_{k-1}(y - \alpha_k), \quad \alpha_k \le y \le d_k;$$
$$z''(y) = g_k(b_{k-1}(y) + \alpha_k) + z_{k-1}(y), \quad 0 \le y \le d_k.$$

(b) For each y, $0 \le y \le d_k$, compute

$$z_k(y) = \begin{cases} z''(y), & \text{if } 0 \le y < \alpha_k \\ \min\{z'(y), z''(y)\}, & \text{if } \alpha_k \le y \le d_k \\ z_k(d_k), & \text{if } d_k < y \le d_n \end{cases}.$$

(c) For each y, $0 \le y \le d_k$, compute

$$a_k(y) = \begin{cases} a_{k-1}(y), & \text{if } 0 \le y < \alpha_k \\ a_{k-1}(y), & \text{if } \alpha_k \le y \le d_k \text{ and } z_k(y) = z''(y) \\ a_{k-1}(y) + \alpha_k, & \text{if } \alpha_k \le y \le d_k \text{ and } z_k(y) = z'(y) \\ a_k(d_k), & \text{if } d_k < y \le d_n \end{cases}.$$

(d) For each y, $0 \le y \le d_k$, compute

$$b_k(y) = \begin{cases} b_{k-1}(y) + \alpha_k, & \text{if } 0 \le y < \alpha_k \\ b_{k-1}(y) + \alpha_k, & \text{if } \alpha_k \le y \le d_k \text{ and } z_k(y) = z''(y) \\ b_{k-1}(y), & \text{if } \alpha_k \le y \le d_k \text{ and } z_k(y) = z'(y) \\ b_k(d_k), & \text{if } d_k < y \le d_n \end{cases}.$$

Step 3. Output $z_n(d_n)$ as the optimal value of the function. The actual take-or-leave decisions (i.e., the values of the decision variables) can be found by backtracking.

Note that each function $z_k(y)$ is monotone non-increasing, since as y grows the problem becomes less constrained, i.e., the more space is available for items to be taken, the less we are forced to make the "leave" decisions. Using similar arguments, note that each function $a_k(y)$ is monotone non-decreasing, since the more space y is available to accommodate the taken items, the more items we may take. Furthermore, note that each function $b_k(y)$ is monotone non-increasing: by a symmetric argument, the more space y is available to accommodate the taken items, the less items we may actually leave. Note also that our DP formulation is univariate and involves three univariate and monotone functions. The time and space needed to solve the recurrences is $O(nd_n)$, i.e., pseudopolynomial in the input size.

4.2. Overview of K-approximation sets and functions

In this section, we provide an overview of the technique of K-approximation sets and functions. In the next section, we adapt the discussed tools to constructing an FPTAS for our problem.

For a function $\varphi : \{A, \ldots, B\} \to \mathbb{R}$ that is not identically zero, denote $\varphi^{\min} := \min_{A \le x \le B}\{|\varphi(x)|: \ \varphi(x) \ne 0\}$, and $\varphi^{\max} := \max_{A \le x \le B}\{|\varphi(x)|\}$.

Halman, Klabjan, Mostagir, Orlin, and Simchi-Levi (2009) have introduced the technique of K-approximation sets and functions, and used it to develop an FPTAS for a certain stochastic inventory control problem. Halman et al. (2014) have applied this tool to develop a framework for transforming rather general classes of stochastic DPs into FPTASs including (i) non-decreasing (respectively, non-increasing) DPs with the single-period cost functions that are non-decreasing (respectively, non-increasing) in the state variable and (ii) convex DPs with the single-period cost functions that have a certain convex structure and the transition function is affine.

This technique has been used to yield FPTASs to various optimization problems, see Halman et al. (2014) and the references therein. Notice that for many of these problems no FPTAS was previously known.

We now present formal definitions related to K-approximation set and functions. Let $K \ge 1$, $\alpha, \tilde{\alpha} \ge 0$ be arbitrary real numbers and let $\varphi, \tilde{\varphi} : \{A, \ldots, B\} \to \mathbb{R}^+$ be arbitrary functions. We say that $\tilde{\alpha}$ is a *K-approximation value of α* if $\alpha \le \tilde{\alpha} \le K\alpha$. We say that $\tilde{\varphi}$ is a *K-approximation function of φ* if $\varphi(x) \le \tilde{\varphi}(x) \le K\varphi(x)$ (i.e., $\tilde{\varphi}(x)$ is a K-approximation value of $\varphi(x)$) for all $x = A, \ldots, B$. Below, we sometime omit the word "value" (respectively, "function") from the term "K -approximation value" (or respectively, "K-approximation function") whenever it is clear from the context.

The following property of K-approximation functions is extracted from Halman et al. (2014, Prop. 5.1), which provides a set of general computational rules of K-approximation functions. Its validity follows directly from the definition of K-approximation functions.

Property 1. *(Calculus of K-approximation functions) (Halman et al., 2014, Prop. 5.1) For $i = 1, 2$ let $K_i \ge 1$, let $\varphi_i, \tilde{\varphi}_i : \{A, \ldots, B\} \to \mathbb{R}^+$ and let $\tilde{\varphi}_i$ be a K_i-approximation of φ_i. Let $\psi_1 : \{A', \ldots, B'\} \to \{A, \ldots, B\}$ be an arbitrary function and $\alpha, \beta \in \mathbb{R}^+$ be arbitrary positive real numbers. The following properties hold:*

- *Linearity of approximation: $\alpha + \beta\tilde{\varphi}_1$ is a K_1 -approximation function of $\alpha + \beta\varphi_1$.*
- *Summation of approximation: $\tilde{\varphi}_1 + \tilde{\varphi}_2$ is a $\max\{K_1, K_2\}$-approximation function of $\varphi_1 + \varphi_2$.*
- *Composition of approximation: $\tilde{\varphi}_1(\psi_1)$ is a K_1 -approximation function of $\varphi(\psi_1)$.*
- *Minimization of approximation: $\min\{\tilde{\varphi}_1, \tilde{\varphi}_2\}$ is a $\max\{K_1, K_2\}$-approximation of $\min\{\varphi_1, \varphi_2\}$.*
- *Maximization of approximation: $\max\{\tilde{\varphi}_1, \tilde{\varphi}_2\}$ is a $\max\{K_1, K_2\}$-approximation of $\max\{\varphi_1, \varphi_2\}$.*
- *Approximation of approximation: If $\varphi_2 = \tilde{\varphi}_1$ then $\tilde{\varphi}_2$ is a K_1K_2-approximation of φ_1.*

Since the problem studied in this paper has a monotone structure over intervals of integer numbers, to simplify the discussion, we concentrate on Halman *et al.*'s definitions for K-approximation sets and functions specialized to monotone functions over intervals of integer numbers. We next turn to defining K-approximation sets. The idea behind such approximation sets is to keep a small (i.e., of a polynomially bounded size) set of points in the domain of a function, ensuring that step interpolation between the function's values on this set guarantees rigorous error bounds.

Definition 1 (Halman et al., 2014, Def. 4.4)**.** Let $\varphi : \{A, \ldots, B\} \to \mathbb{R}$ be a monotone function. For any subset $W \subseteq \{A, \ldots, B\}$ satisfying $A, B \in W$, the *approximation of φ induced by W* is the function

$$\hat{\varphi}(x) = \begin{cases} \varphi(\min_{y \in W}\{y \geq x\}), & \text{if } \varphi \text{ is a non-decreasing function,} \\ \varphi(\max_{y \in W}\{y \leq x\}), & \text{if } \varphi \text{ is a non-increasing function.} \end{cases}$$

Definition 2 (Halman et al., 2014, Def. 4.2 and Prop. 4.5)**.** Let $K \geq 1$ and let $\varphi : \{A, \ldots, B\} \to \mathbb{R}^+$ be a monotone function. Let $W \subseteq \{A, \ldots, B\}$ be a subset satisfying $A, B \in W$. We say that W is a *K-approximation set of φ* if the approximation of φ induced by W is a K-approximation function of φ.

In all algorithms discussed in this section, we assume that for an input function φ an oracle is available which for any x returns the value $\varphi(x)$ in t_φ time. The statement below asserts that for a monotone function φ a K-approximation set of a polynomial size can be found in polynomial time.

Proposition 1. *(Halman et al., 2014, Prp. 4.6) Let $\varphi : \{A, \ldots, B\} \to \mathbb{R}^+$ be a monotone function, for which an oracle with a query time of t_φ is available. Then for every $K > 1$, it is possible to compute a K-approximation set of φ of size $O(\log_K \frac{\varphi^{\max}}{\varphi^{\min}})$ in $O\big(t_\varphi(1 + \log_K \frac{\varphi^{\max}}{\varphi^{\min}}) \log(B - A)\big)$ time.*

A procedure for constructing a K-approximation function for any monotone function $\varphi : \{A, \ldots, B\} \to \mathbb{R}^+$ is stated as Function Compress.

Function Compress

Inputs: $\varphi, \{A, \ldots, B\}, K$, where $\varphi : \{A, \ldots, B\} \to \mathbb{R}^+$ is a monotone function represented by an appropriate oracle

Returns:a monotone K-approximation of φ

Step 1. Obtain a K-approximation set W of φ over the domain of $\{A, \ldots, B\}$.

Step 2. Return $\tilde{\varphi}$, the approximation of φ induced by W as an array$\{(x, \tilde{\varphi}(x)) | x \in W\}$ sorted in increasing order of x.

As demonstrated by Halman et al. (2014, Prop. 4.5), K-approximations of a function φ can be found even if the function itself is not available, but there exists an oracle that computes values of some function $\bar{\varphi}$ that is an approximation function of φ. Below we present a statement, adapted from Halman et al. (2014, Prop. 4.5), that applies to finding approximations of a monotone function φ by calling Function Compress.

Proposition 2. *Let $K_1, K_2 \geq 1$ be real numbers and let $\varphi : \{A, \ldots, B\} \to \mathbb{R}^+$ be a monotone function. Let $\bar{\varphi}$ be a monotone K_2-approximation function of φ. Then Function Compress$(\bar{\varphi}, \{A, \ldots, B\}, K_1)$ returns in $O\big(t_\varphi(1 + \log_{K_1} \frac{\varphi^{\max}}{\varphi^{\min}}) \log(B - A)\big)$ time a monotone step function $\tilde{\varphi}$ with $O(\log_{K_1} \frac{\varphi^{\max}}{\varphi^{\min}})$ steps that $K_1 K_2$ -approximates φ, and of which the query time is $t_{\tilde{\varphi}} = O(\log \log_{K_1} \frac{\varphi^{\max}}{\varphi^{\min}})$.*

In Proposition 2, the estimates of computation times and approximation quality follow from the discussion above and an application of the calculus of approximation (the approximation of approximation rule).

4.3. FPTAS design and analysis

We now develop and analyze an FPTAS for problem (5). For our FPTAS to work, we need certain assumptions regarding properties of the non-decreasing functions f_j and g_j. The first assumption is identical to the one used in Section 3, i.e., there exists a constant $r \geq 1$ such that for any positive u inequality (15) holds. The second assumption is that the following similar inequality

$$f_j(ux) \leq u^r f_j(x) \tag{24}$$

holds for each j, $1 \leq j \leq n$, and for any positive u. Assumption (15) (respectively, (24)) tells us that if $\tilde{x}$ is a K-approximation value of x then $g_j(\tilde{x})$ (respectively, $f_j(\tilde{x})$) is a K^r-approximation of $g_j(x)$ (respectively, of $f_j(x)$). Notice that the assumption (24) is used for convenience in the beginning of our reasoning and is later dropped.

Our FPTAS is based on Algorithm DPab given in Section 4.1. In every iteration k, we obtain $\tilde{z}_k(\cdot)$, $\tilde{a}_k(\cdot)$ and $\tilde{b}_k(\cdot)$ that are approximation functions of the true functions $z_k(\cdot)$, $a_k(\cdot)$ and $b_k(\cdot)$, respectively. Each of these approximate functions is obtained by calling Function Compress with the relevant input. The input functions $\bar{z}_k(\cdot), \bar{a}_k(\cdot)$ and $\bar{b}_k(\cdot)$ used in these calls are represented by the corresponding oracles built in line with Algorithm DPab, and are, in turn, approximation functions of the true functions $z_k(\cdot)$, $a_k(\cdot)$ and $b_k(\cdot)$, respectively. Due to the calls of Compress in Step 3(b) and 3(c), with each iteration the quality of approximation of each $\tilde{a}_k(\cdot)$ and $\tilde{b}_k(\cdot)$ deteriorates by a factor of K compared to $\tilde{a}_{k-1}(\cdot)$ and $\tilde{b}_{k-1}(\cdot)$, respectively. The approximation of $\tilde{z}_k(\cdot)$ deteriorates by a factor of K^{r+1} due to the call of Compress in Step 3(a) coupled with the fact that when computing each function $f_k(\cdot)$ and $g_k(\cdot)$ with an argument that is a K-approximation value of the true argument, the calculated value is a K^r-approximation value of the true value; see the discussion in the previous paragraph.

Formally, the approximation scheme can be stated as follows.

Algorithm EpsMin2

Step 1. Compute Z^{UB} by (16). For a given positive ε, compute $K = (1 + \varepsilon)^{\frac{1}{r(n-1)+n}}$.

Step 2. Start with the initial states $\big(0, \tilde{z}_0(y), \tilde{a}_0(y), \tilde{b}_0(y)\big)$, where each function is represented as a two-component array compatible with the structure of the output of Function Compress, i.e., by ((0, 0), $(d_n, 0)$).

Step 3. For all k from 1 to n, do

(a) Determine function $\tilde{z}_k(\cdot)$ returned by calling Function Compress$(\bar{z}_k(\cdot), \{0, \ldots, d_n\}, K)$, provided that for computing values of the input function $\bar{z}_k(\cdot)$ the following oracle is used:

$$\bar{z}_k(y) = \begin{cases} z''(y), & \text{if } 0 \leq y < \alpha_k \\ \min\{z'(y), z''(y)\}, & \text{if } \alpha_k \leq y \leq d_k, \\ \bar{z}_k(d_k), & \text{if } d_k < y \leq d_n \end{cases}$$

where

$$z'(y) = f_k(\tilde{a}_{k-1}(y) + \alpha_k) + \tilde{z}_{k-1}(y - \alpha_k);$$

$$z''(y) = g_k(\tilde{b}_{k-1}(y) + \alpha_k) + \tilde{z}_{k-1}(y).$$

(b) Determine function $\tilde{a}_k(\cdot)$ returned by calling Function Compress$(\bar{a}_k(\cdot), \{0, \ldots, d_n\}, K)$, provided that for computing values of the input function $\bar{a}_k(\cdot)$ the following oracle is used:

$$\bar{a}_k(y) = \begin{cases} \tilde{a}_{k-1}(y), & \text{if } 0 \leq y < \alpha_k \\ \tilde{a}_{k-1}(y), & \text{if } \alpha_k \leq y \leq d_k \text{ and } \bar{z}_k(y) = z''(y) \\ \tilde{a}_{k-1}(y) + \alpha_k, & \text{if } \alpha_k \leq y \leq d_k \text{ and } \bar{z}_k(y) = z'(y) \\ \bar{a}_k(d_k), & \text{if } d_k < y \leq d_n \end{cases}.$$

(c) Determine function $\tilde{b}_k(\cdot)$ returned by calling Function Compress$\big(\bar{b}_k(\cdot), \{0, \ldots, d_n\}, K\big)$, provided that for computing values of the input function $\bar{b}_k(\cdot)$ the following oracle is used:

$$\bar{b}_k(y) = \begin{cases} \tilde{b}_{k-1}(y) + \alpha_k, & \text{if } 0 \leq y < \alpha_k \\ \tilde{b}_{k-1}(y) + \alpha_k, & \text{if } \alpha_k \leq y \leq d_k \text{ and } \bar{z}_k(y) = z''(y) \\ \tilde{b}_{k-1}(y), & \text{if } \alpha_k \leq y \leq d_k \text{ and } \bar{z}_k(y) = z'(y) \\ \bar{b}_k(d_k), & \text{if } d_k < y \leq d_n \end{cases}.$$

Step 4. Output $\tilde{z}_n(d_n)$ as an approximate value of the smallest value $z_n(d_n)$ of the original objective. The actual take-or-leave decisions (i.e., the values of the decision variables) can be found by backtracking.

We now analyze the performance of Algorithm EpsMin2.

Lemma 2. *If applied to problem (5) that satisfies (15) and (24), Algorithm EpsMin2 for each j, $0 \le j \le n$, computes functions such that*

- *$\tilde{z}_j(\cdot)$ is a $K^{r\max\{j-1,0\}+j}$-approximation of $z_j(\cdot)$;*
- *$\tilde{a}_j(\cdot)$ and $\tilde{b}_j(\cdot)$ is a K^j -approximation of $a_j(\cdot)$ and of $b_j(\cdot)$, respectively.*

Proof. We notice first that $\tilde{z}_k(\cdot), \tilde{a}_k(\cdot), \tilde{b}_k(\cdot)$ are all monotone functions for each k, $0 \le k \le n$. Therefore, all calls to Compress are well defined and we may use Proposition 2 in our analysis.

For some q, $1 < q < n$, assume that the lemma holds for each j, $1 \le j \le q-1 < n$. In particular, we assume that

- $\tilde{z}_{q-1}(\cdot)$ is a $K^{r(q-2)+q-1}$-approximation of $z_{q-1}(\cdot)$;
- $\tilde{a}_{q-1}(\cdot)$ and $\tilde{b}_{q-1}(\cdot)$ is a K^{q-1} -approximation of $a_{q-1}(\cdot)$ and of $b_{q-1}(\cdot)$, respectively.

We want to prove that

- $\tilde{z}_q(\cdot)$ is a $K^{r(q-1)+q}$-approximation of $z_q(\cdot)$;
- $\tilde{a}_q(\cdot)$ and $\tilde{b}_q(\cdot)$ is a K^q -approximation of $a_q(\cdot)$ and of $b_q(\cdot)$, respectively.

Running Step 3(b) of Algorithm EpsMin2 for $k = q$, we get by the induction hypothesis and the calculus of approximation (the linearity of approximation rule) that $\bar{a}_q(\cdot)$ is a K^{q-1}-approximation of $a_q(\cdot)$. Looking at the output of Function Compress$\left(\bar{a}_q(\cdot), \{0,\ldots,d_n\}, K\right)$, we deduce from Proposition 2 applied with $K_1 = K$ and $K_2 = K^{q-1}$, that $\tilde{a}_q(\cdot)$ is a K^q-approximation of $a_q(\cdot)$, as required. Similarly, considering Step 3(c) of Algorithm EpsMin2, we deduce that $\tilde{b}_q(\cdot)$ is a K^q -approximation of $b_q(\cdot)$.

We now turn to evaluating the approximation ratio of $\tilde{z}_q(\cdot)$. By the induction hypothesis and the calculus of approximation (the linearity of approximation rule), we get that $\tilde{b}_{q-1}(y)+\alpha_j$ is a K^{q-1}-approximation of $b_{q-1}(y)+\alpha_j$. Therefore, it follows from (15) that $g_q(\tilde{b}_{q-1}(y)+\alpha_j)$ is a $K^{r(q-1)}$-approximation of $g_q(b_{q-1}(y)+\alpha_j)$. Using the induction hypothesis for $\tilde{z}_{q-1}$ and the calculus of approximation (the summation of approximation rule), we get that $g_q(\tilde{b}_{q-1}(y)+\alpha_j)+\tilde{z}_{q-1}(y)$ is a $K^{\max\{r(q-1),r(q-2)+q-1\}}$-approximation and, therefore, is a $K^{r(q-1)+q-1}$-approximation of $g_q(b_{q-1}(y)+\alpha_j)+z_{q-1}(y)$. Similarly, using (24), the composition of approximation rule and the summation of approximation rule, we obtain that $f_q(\tilde{a}_{q-1}(y)+\alpha_j)+\tilde{z}_{q-1}(y-\alpha_j)$ is a $K^{r(q-1)+q-1}$ -approximation of $f_q(a_{q-1}(y)+\alpha_j)+z_{q-1}(y-\alpha_j)$. Using once more the calculus of approximation (the minimization of approximation rule) we derive that $\bar{z}_q(y)$ is a $K^{r(q-1)+q-1}$-approximation of $z_q(y)$. Looking at the output of Function Compress$\left(\bar{z}_q(\cdot), \{0,\ldots,d_n\}, K\right)$, we deduce from Proposition 2 applied with $K_1 = K$ and $K_2 = K^{r(q-1)+q-1}$, that $\tilde{z}_q(\cdot)$ is a $K^{r(q-1)+q}$-approximation of $z_q(\cdot)$, as required. □

Theorem 2. *For problem (5) that satisfies (15) and (24), Algorithm EpsMin2 is an FPTAS that computes a $(1+\varepsilon)$-approximation value of $z_n(d_n)$ in*

$$O\left(\frac{n^2(\log Z^{UB}+\log A_n)\log d_n}{\varepsilon}\left(\log\frac{n\log Z^{UB}}{\varepsilon}+\log\frac{n\log A_n}{\varepsilon}\right)\right)$$

time.

Proof. Lemma 2 implies that $\tilde{z}_n(d_n)$ is a $K^{r(n-1)+n}$-approximation value of the optimal value $z_n(d_n)$ of the objective function. For K defined as in Step 1 of Algorithm EpsMin2, we obtain that $\tilde{z}_n(d_n) \le (1+\varepsilon)z_n(d_n)$, which provides the desired accuracy.

Now, we turn to analyzing the running time of the algorithm. Steps 1 and 2 require constant time. Note that Step 2 defines oracles to retrieve the zero values $\tilde{z}_0(y), \tilde{a}_0(y)$ and $\tilde{b}_0(y)$ for y values that are required in computation in Step 3 for $k = 1$.

For any k, $1 \le k \le n$, the largest value that function $\tilde{z}_k(y)$ may achieve is Z^{UB} and its domain is $\{0,\ldots,d_n\}$. Thus, Proposition 2 implies that the running time of Step 3(a) for a fixed k is $O\left(t_{\tilde{z}_k}\log_K Z^{UB}\log d_n\right)$. Using the oracle defined for function $\tilde{z}_k(\cdot)$, we have that $t_{\tilde{z}_k} = O(t_{\tilde{z}_{k-1}} + t_{\tilde{a}_{k-1}} + t_{\tilde{b}_{k-1}})$. It follows from Proposition 2 applied with $K_1 = K$ that $t_{\tilde{a}_{k-1}} = t_{\tilde{b}_{k-1}} = O(\log\log_K A_n)$, since the largest value that each function $\tilde{a}_q(\cdot)$ and $\tilde{b}_q(\cdot)$ may achieve is A_n, i.e., the sum of the weights of all items. Similarly, $t_{\tilde{z}_{k-1}} = O(\log\log_K Z^{UB})$, so that

$$t_{\tilde{z}_k} = O(\log\log_K Z^{UB} + \log\log_K A_n). \tag{25}$$

The running time of Step 3(b) for a fixed k is $O\left(t_{\tilde{a}_k}\log_K A_n\log d_n\right)$, and the structure of the corresponding oracle implies that

$$t_{\tilde{a}_k} = O(t_{\tilde{z}_k} + t_{\tilde{a}_{k-1}}) = O(\log\log_K Z^{UB} + \log\log_K A_n). \tag{26}$$

By symmetry, the running time of Step 3(c) for a fixed k is $O\left(t_{\tilde{b}_k}\log_K A_n\log d_n\right)$, where $t_{\tilde{b}_k} = O(t_{\tilde{a}_k})$.

We enter Step 4 having found the array representation of function $\tilde{z}_n(\cdot)$. By Proposition 2 it takes $O(\log\log_K Z^{UB}) = O\left(\log\frac{n\log Z^{UB}}{\varepsilon}\right)$ time to compute $\tilde{z}_n(y)$ for any value y (i.e., the dependency on n is only logarithmic). In order to build the feasible solution that $\tilde{z}_n(x)$ approximates, we need to perform backtracking to discover the various values of the n leave-or-take decisions and then re-evaluate z_n, exactly as is done in Step 3 of Algorithm EpsMin1. This additionally takes $O(n)$ time.

Thus, the running time of Algorithm EpsMin2 is determined by the total time complexity of Step 3 over all iterations, which is $O(n\left(\log_K Z^{UB}+\log_K A_n\right)\log d_n\left(\log\log_K Z^{UB}+\log\log_K A_n\right))$. Moving to the base 2 logarithms, using the equation $\log K = \log\sqrt[r(n-1)+n]{1+\varepsilon} = O(\frac{\varepsilon}{rn})$ and taking into account that r is constant, the claimed running time follows. □

As reflected in its name, Algorithm DPab computes both values $a_k(y)$ and $b_k(y)$. Notice that the value $a_k(y)$ is closely related to y_k, introduced in Section Section 3, since both of them represent the total weight of the taken items after items $1,\ldots,k$ have been considered; the difference is that there is a space limit y in the case of $a_k(y)$. Similarly, $b_k(y)$ is closely related to $\hat{y}_k$.

5. Extensions

In this section, we extend the approach described earlier in this paper to designing an FPTAS for the same problem without assuming that both conditions (15) and (24) hold (Section 5.1), and for problems that are generalizations or variations of problem (5) (Sections 5.2–5.4).

It turns out that the method of K-approximation sets and functions is flexible enough, so that only minor adjustments are required to handle these variations. As a rule, we only state the changes that are needed in the corresponding DP algorithm, while its conversion to an FPTAS can be done quite similarly to Algorithm EpsMin2. In Sections 5.2–5.4 it is assumed that both conditions (15) and (24) hold; if required, one of them can be removed as described in Section 5.1.

5.1. Only one of the conditions (15) and (24) holds

If we want to design an FPTAS for problem (5) for which the condition (24) is dropped, we need to compute the quantities $a_k(y)$ not directly as done in Algorithm DPab, but express them in terms

of $b_k(y)$. This observation leads to the following modified DP algorithm, which we call Algorithm DPb, since it computes values $b_k(y)$ only.

Algorithm DPb

Step 1. Start with the initial states $(0, z_0(y), b_0(y))$, $0 \le y \le d_n$, defined by

$$z_0(y) = b_0(y) = 0, \quad y = 0, \ldots, d_n.$$

Step 2. For all k from 1 to n find the states $(k, z_k(y), b_k(y))$, as follows:

(a) Compute

$$z'(y) = f_k(A_k - b_{k-1}(y)) + z_{k-1}(y - \alpha_k), \ \alpha_k \le y \le d_k;$$
$$z''(y) = g_k(b_{k-1}(y) + \alpha_k) + z_{k-1}(y), \ 0 \le y \le d_k.$$

(b) For each y, $0 \le y \le d_k$, compute

$$z_k(y) = \begin{cases} z''(y), & \text{if } 0 \le y < \alpha_k \\ \min\{z'(y), z''(y)\}, & \text{if } \alpha_k \le y \le d_k \\ z_k(d_k), & \text{if } d_k < y \le d_n \end{cases}.$$

(c) For each y, $0 \le y \le d_k$, compute

$$b_k(y) = \begin{cases} b_{k-1}(y) + \alpha_k, & \text{if } 0 \le y < \alpha_k \\ b_{k-1}(y) + \alpha_k, & \text{if } \alpha_k \le y \le d_k \text{ and } z_k(y) = z''(y) \\ b_{k-1}(y), & \text{if } \alpha_k \le y \le d_k \text{ and } z_k(y) = z'(y) \\ b_k(d_k), & \text{if } d_k < y \le d_n \end{cases}.$$

Step 3. Output $z_n(d_n)$ as the optimal value of the function. The actual take-or-leave decisions (i.e., the values of the decision variables) can be found by backtracking.

Algorithm EpsMin2 should be modified accordingly, i.e., in Step 1 we also need to compute the values $A_k, k = 1, 2, \ldots, n$, by (10), in Step 2 $\tilde{a}_0(y)$ has to be removed, Step 3(b) has to be removed all together and the formula for $z'(y)$ in Step 3(a) has to become

$$z'(y) = f_k(A_k - \tilde{b}_{k-1}(y)) + \tilde{z}_{k-1}(y - \alpha_k), \quad \alpha_k \le y \le d_k.$$

The analysis of the accuracy of the resulting scheme is similar to that in the proof of Lemma 2. However, we should be aware that the fact that $\tilde{b}_{q-1}(y)$ is a K^{q-1}-approximation of $b_{q-1}(y)$ does not imply that $A_q - \tilde{b}_{q-1}(y)$ is a K^{q-1}-approximation of $A_q - b_{q-1}(y)$. Nevertheless, it follows that $A_q - \tilde{b}_{q-1}(y) \le A_q - b_{q-1}(y)$, so that the right value is appropriately approximated.

In Step 4 of the modified scheme, finding the value of $\tilde{z}_n(y)$ will take time that is linear in n, not logarithmic in n, as in the proof of Theorem 2. The reason is that while the value of the resulting $\tilde{z}_n(d_n)$ is assured to be bounded by $(1+\varepsilon)$ times the optimal value Z_n^*, it may be below it. Therefore, once an approximated value $\tilde{z}_n(d_n)$ is found, we must perform backtracking in order to identify "take-or-leave" decisions that lead to a feasible solution corresponding to $\tilde{z}_n(d_n)$ and then calculate the exact value of this feasible solution using the previously-identified "take-or-leave" decisions.

If we want to design an FPTAS for problem (5) for which the condition (15) is dropped, we need to compute the quantities $b_k(y)$ not directly as done in Algorithm DPab, but express them in terms of $a_k(y)$. This observation leads to a modified DP algorithm, which we call Algorithm DPa, since it computes values $a_k(y)$ only. For the sake of brevity, we will not state the algorithms but instead outline the changes needed to be performed in Algorithm DPab and EpsMin2. In Algorithm DPab we delete all references to function $b_k(\cdot)$. In Step 2(a) the formula for $z''(y)$ has to become

$$z''(y) = g_k(A_k - a_{k-1}(y)) + z_{k-1}(y), \quad 0 \le y \le d_k.$$

We also drop Step 2(d) all together. Algorithm EpsMin2 should be modified accordingly, i.e., in Step 1 we also need to compute the values $A_k, k = 1, 2, \ldots, n$, by (10), in Step 2 $\tilde{b}_0(y)$ has to be removed, Step 3(c) has to be removed all together and the formula for $z''(y)$ in Step 3(a) has to become

$$z''(y) = g_k(A_k - \tilde{a}_{k-1}(y)) + \tilde{z}_{k-1}(y), \quad 0 \le y \le d_k.$$

Thus, the performed modifications do not alter neither the accuracy of the approximation scheme, nor its running time, and the following statement holds.

Theorem 3. *For problem (5) that satisfies only one of the conditions (15) and (24), there exists an FPTAS that takes*

$$O\left(\frac{n^2(\log Z^{UB} + \log A_n)\log d_n}{\varepsilon}\left(\log\frac{n\log Z^{UB}}{\varepsilon} + \log\frac{n\log A_n}{\varepsilon}\right)\right)$$

time.

We note that all three algorithms DPab, DPa, and DPb are primal DP algorithms with respect to the state variable y that represents an available space y.

5.2. Generalized objective function

Consider the problem that differs from problem (5) in the additional terms $\sum_{j=1}^n \varphi_j(x_j) + \sum_{j=1}^n \psi_j(1 - x_j)$ added to the objective function, where all φ_j and ψ_j are non-negative and non-decreasing functions of the binary variables x_j. To modify Algorithm DPab to attend this change, we only need to replace the computation in Step 2(a) by the following

Step 2(a′). Compute

$$z'(y) = f_k(a_{k-1}(y) + \alpha_k) + z_{k-1}(y - \alpha_k) + \varphi_j(1) + \psi_j(0), \quad \alpha_k \le y \le d_k;$$
$$z''(y) = g_k(b_{k-1}(y) + \alpha_k) + z_{k-1}(y) + \varphi_j(0) + \psi_j(1), \quad 0 \le y \le d_k.$$

The Steps 2(b)–(d) remain the same, provided that $z'(y)$ and $z''(y)$ are those computed in Step 2(a′) above. The boundary condition remains the same as in Step 1 of Algorithm DPab.

5.3. Generalized nested constraints

Similarly, we can handle an extension of problem (5) in which different coefficients α_j and β_j are involved in the lines of constraints and the objective function, respectively. The extended problem can stated as

$$\text{Minimize} \quad Z(\mathbf{x}) = \sum_{j=1}^n f_j\left(\sum_{i=1}^j \beta_i x_i\right)x_j + \sum_{j=1}^n g_j\left(\sum_{i=1}^j \beta_i(1-x_i)\right)(1-x_j),$$

$$\text{Subject to} \quad \sum_{j=1}^k \alpha_j x_j \le d_k, \quad 1 \le k \le n,$$
$$x_j \in \{0,1\}, \quad j = 1, \ldots, n,$$

Similarly to the above, to obtain an updated DP algorithm for solving this problem, we only need to replace the computation in Step 2(a) by the following

Step 2(a″). Compute

$$z'(y) = f_k(a_{k-1}(y) + \beta_k) + z_{k-1}(y - \alpha_k), \ \alpha_k \le y \le d_k;$$
$$z''(y) = g_k(b_{k-1}(y) + \beta_k) + z_{k-1}(y), \ 0 \le y \le d_k.$$

The Steps 2(b)–(d) remain the same, provided that $z'(y)$ and $z''(y)$ are those computed in Step 2(a″) above. The boundary condition remains the same.

5.4. Maximization problem

In this section, we show that principles similar to those presented earlier in this paper can be used to develop an FPTAS for the maximization variant of problem (5), i.e., the problem

$$\text{Maximize } Z=\sum_{j=1}^{n} f_j\left(\sum_{i=1}^{j}\alpha_i x_i\right)x_j+\sum_{j=1}^{n} g_j\left(\sum_{i=1}^{j}\alpha_i(1-x_i)\right)(1-x_j)$$

$$\text{Subject to } \sum_{j=1}^{k}\alpha_j x_j \le d_k,\ \ 1\le k\le n,$$

$$x_j\in\{0,1\},\ \ j=1,2,\ldots,n. \tag{27}$$

Recall that for the maximization problem, the definition of an FPTAS has to be adjusted. For a vector of decision variables $\mathbf{x}$, consider a problem of maximizing a function $\varphi(\mathbf{x})$ that takes positive values. An FPTAS finds a feasible solution $\mathbf{x}^H$ such that for a small positive ε the inequality $\varphi(\mathbf{x}^H)\ge(1-\varepsilon)\varphi(\mathbf{x}^*)$ holds, and the required running time is polynomial with respect to the length of the problem's input and $1/\varepsilon$.

Notice that in combinatorial optimization the approximability issues are not symmetric if one switches from a minimization problem to its maximization counterpart. First, it is possible that the minimization problem is polynomially solvable, while its maximization analogue is NP-hard. The best known example of this type is the min-cut vs the max-cut problem. Another example is the problem of minimizing of the half-product function (4) with no additional constraints, which is NP-hard (Badics & Boros, 1998), while its maximization counterpart is solvable in $O(n^3)$ time, as shown in Kellerer, Sarto Basso, and Strusevich (2017).

Second, it is possible that the minimization problem admits an approximation algorithm or even an FPTAS, while the maximization version is not approximable. Such examples can be found even within the range of problems of immediate interest to this paper. The problem of minimizing the half-product function (4) subject to the knapsack constraint (2) admits an FPTAS that requires $O(n^2/\varepsilon)$ time (Sarto Basso & Strusevich, 2017), while its maximization counterpart does not admit a constant ratio approximation algorithm unless $P\neq NP$, as proved in Kellerer et al. (2017).

Still, in the case under consideration an FPTAS for problem (27) is fairly easy to derive from the same principles as either Algorithm EpsMin1 or Algorithm EpsMin2.

5.4.1. Geometric rounding approach

We will apply the same principles as Algorithm EpsMin1. This is done by converting a primal version of the DP algorithm, which is presented below. Such an algorithm uses state variables y_k, which denote the total weight of taken items j, $1\le j\le k$.

Although the primal DP algorithm is very similar to Algorithm DDP presented in Section 3, to avoid ambiguity we provide its formal description below.

Algorithm PDP

Step 1. Start with the initial state $(0,Z_0,y_0)=(0,0,0)$. Compute the values A_k, $k=1,2,\ldots,n$, by (10).

Step 2. For all k from 0 to $n-1$ do

Make transitions from each stored primal state of the form $(k,\ Z_k,\ y_k)$ into the states of the form $\left(k+1,Z_{k+1},y_{k+1}\right)$ by assigning the next variable x_{k+1}.

(a) Define $x_{k+1}=1$, provided that it is feasible to take item $k+1$, i.e., if the $(k+1)$-th nested constraint $y_k+\alpha_{k+1}\le d_{k+1}$ holds. If feasible, the assignment $x_{k+1}=1$ creates a state of the form $\left(k+1,Z_{k+1},y_{k+1}\right)$, where $y_{k+1}=y_k+\alpha_{k+1};\ Z_{k+1}=Z_k+f_{k+1}\left(y_{k+1}\right)$.

(b) Define $x_{k+1}=0$, which is always feasible. This assignment creates a state of the form $\left(k+1,Z_{k+1},y_{k+1}\right)$, where $y_{k+1}=y_k;\ Z_{k+1}=Z_k+g_{k+1}\left(A_{k+1}-y_{k+1}\right)$.

Step 3. Output the optimal value of the function that corresponds to the largest value of Z_n among all found states of the form $(n,\ Z_n,\ y_n)$.

Similarly to Section 3, in order to convert this DP algorithm to an FPTAS, we assume that each function f_j and g_j, $1\le j\le n$, can be computed in constant time. Additionally, we assume that the condition (24) holds.

An FPTAS for problem (27) uses similar principles as Algorithm EpsMin1. We refer to the resulting scheme as Algorithm EpsMax. Its Step 1 is identical to the one of Algorithm EpsMin1 with the exception of changing the single occurrence of $\hat{y}$ to y. The remaining two steps are:

Step 2. Store the initial state (0,0,0). For each k, $0\le k\le n-1$, do the following:

According to Algorithm PDP move from a stored primal state $(k,\ Z_k,\ y_k)$ to at most two primal states of the form $(k+1,Z_{k+1},y_{k+1})$, where $Z_{k+1}\le Z^{UB}$. If the number of generated states $(k+1,Z_{k+1},y_{k+1})$ with the Z-values in the same interval J_t and with the y-values in the same interval I_ℓ exceeds one, then keep only one of these states, that with the smallest y -value.

Step 3. Among all values Z_n found in Step 2 identify the largest one. With this value of Z_n, perform the backtracking to find the corresponding decision variables x_j, $j=1,\ldots,n$. Compute the value of the objective function with the found x_j's, call this value Z^ε and accept it as an approximate value of the objective function.

The following statement holds for the output of Algorithm EpsMax.

Lemma 3. *Assume that the dynamic programming Algorithm PDP is applied to problem (27) and finds a chain of primal states*

$$(0,0,0),(1,Z_1^*,y_1^*),\ldots,(n,Z_n^*,y_n^*)$$

leading to the optimal value $Z^*=Z_n^*$. Then for each k, $1\le k\le n$, Algorithm EpsMax finds a state $(k,\ Z_k,\ y_k)$ such that

$$y_k\le y_k^*\le(1+\varepsilon)^{\frac{k}{n}}y_k \tag{28}$$

and

$$(1+\varepsilon)^{\frac{k}{n}}Z_k\ge Z_k^*. \tag{29}$$

The proof of Lemma 3 is symmetric to that of Lemma 1 and is therefore omitted. The running time of Algorithm EpsMax is the same as that of Algorithm EpsMin1. To verify the accuracy of Algorithm EpsMax, notice that it outputs a feasible state $(n,\ Z_n,\ y_n)$ such that (29) holds for $k=n$, i.e.,

$$(1+\varepsilon)Z_n\ge Z_n^*,$$

which implies that

$$Z_n>\left(1-\varepsilon^2\right)Z_n\ge(1-\varepsilon)Z_n^*,$$

as required by the definition of an FPTAS for a maximization problem.

5.4.2. K-approximation sets and functions approach

The technique of K-approximation sets and functions also applies to maximization problems. However, for the maximization problems the definitions presented in Section 4.2 have to be adjusted, as described, i.e., in Section 10.1 of Halman et al. (2014).

For the minimization problems, we have considered the one-sided approximation, where for every $K\ge1$ we construct a function

$\tilde{z}$ that K-approximates z, i.e., $z(x) \le \tilde{z}(x) \le Kz(x)$, for every x. It is convenient to say that $\tilde{z}$ is a K-approximation of z *from above*. For the maximization problems, we would like to construct an approximation function $\tilde{z}$ so that the error remains one-sided but is on the other side. In other words, $\tilde{z}$ is said to be a K-approximation of z *from below* if $\frac{z}{K} \le \tilde{z} \le z$. Clearly, if $\tilde{z}$ K-approximates z from above, then $\frac{\tilde{z}}{K}$ K-approximates z from below. Similarly, if $\tilde{z}$ K-approximates z from below, then $K\tilde{z}$ K-approximates z from above.

In the following, we describe the changes needed in Algorithms DPab and EpsMin2 in order to approximate the maximization problem (27). In Step 2(b) of Algorithm DPab we change the single occurrence of "min" to "max". We notice that thus each function $z_k(y)$ becomes monotone non-decreasing (as opposed to non-increasing in the minimization problem (5)), since as y grows the problem becomes less constrained, i.e., the more space is available for items to be taken, the less we are forced to make the "leave" decision. We also notice that each function $a_k(y)$ is monotone non-decreasing and each function $b_k(y)$ is monotone non-increasing, exactly as in the minimization problem (5). Since all functions $z_k(y)$, $a_k(y)$, $b_k(y)$ are monotone, we can still apply the K -approximation sets and functions technique outlined in Section 4.2. The changes required in Algorithm EpsMin2 are the following. In Step 3(a) we change the single occurrence of "min" to "max". In Step 4 we output $\frac{\tilde{z}_n(d_n)}{1+\varepsilon}$ (instead of $\tilde{z}_n(d_n)$). The proof of Lemma 2 remains the same with the exception of using the maximization of approximation rule (as opposed to minimization of approximation rule). The proof of Theorem 2 remains the same.

6. Conclusion

We consider the problem of Boolean programming with a non-separable non-linear objective function that reflects the take-or-leave decisions to be made regarding n available items. We present several examples of practical situations in which such a problem arises. We report two approaches to developing an FPTAS based on converting a DP algorithm by the use of the geometric rounding technique and by adapting the K -approximation sets and functions technique. The running times of the resulting approximation schemes compare favorably with known analogues for less general problems.

The FPTAS given in Section 3 uses a dual DP formulation and is based only on geometric rounding. The FPTAS given in Section 4 uses a primal DP formulation and the technique of K -approximation sets and functions. While the latter FPTAS uses less elementary methods, it runs faster with respect to both the number of items n and the relative error ε, and can be relatively easily adjusted to cope with more general problems, as demonstrated in Section 5.

It is remains to be seen whether the approaches developed in Woeginger (2000) and Kovalyov and Kubiak (2012) can lead to faster FPTASs than the best ones presented in this paper.

Notice that both approaches require additional assumptions on the rate of growth of either functions f_k or g_k. It is interesting to point out, that despite the difference in the applied approaches, the same conditions given either by (24) or by (15) are introduced. Although theses assumptions reduce an applicability range of the studied models, still they are satisfied by the polynomial functions, which is a quite representative class of objectives; see, e.g., the survey Hochbaum (2007). It is an interesting research goal to design an FPTAS for problem (5), provided that functions f_k and g_k are general monotone non-decreasing (i.e., without assuming (24) or (15)) or to establish that such a general version of the problem does not admit an FPTAS.

References

Badics, T., & Boros, E. (1998). Minimization of half-products. *Mathematics of Operations Research, 33*, 649–660.

Brahimi, N., Absi, N., Dauzère-Pérès, S., & Nordli, A. (2017). Single-item dynamic lot–sizing problems: An updated survey. *European Journal of Operational Research, 263*, 838–863.

Chubanov, S., Kovalyov, M., & Pesch, E. (2006). An FPTAS for a single-item capacitated economic lot-sizing problem with monotone cost structure. *Mathematical Programming A, 106*, 453–466.

Chubanov, S., Kovalyov, M., & Pesch, E. (2008). A single-item economic lot-sizing problem with a non-uniform resource: Approximation. *European Journal of Operational Research, 189*, 877–889.

Erel, E., & Ghosh, J. B. (2008). FPTAS for half-products minimization with scheduling applications. *Discrete Applied Mathematics, 156*, 3046–3056.

Goyal, V., Genc-Kaya, L., & Ravi, R. (2011). An FPTAS for minimizing the product of two non-negative linear cost functions. *Mathematical Programming, 126*, 401–405.

Goyal, V., & Ravi, R. (2013). An FPTAS for minimizing a class of low-rank quasi–concave functions over a convex set. *Operations Research Letters, 41*, 191–196.

Halman, N., Klabjan, D., Li, C. L., Orlin, J., & Simchi-Levi, D. (2014). Fully polynomial time approximation schemes for stochastic dynamic programming. *SIAM Journal on Discrete Mathematics, 28*, 1725–1796.

Halman, N., Klabjan, D., Mostagir, M., Orlin, J., & Simchi-Levi, D. (2009). A fully polynomial time approximation scheme for single-item stochastic inventory control with discrete demand. *Mathematics of Operations Research, 34*, 674–685.

Halman, N., Orlin, J., & Simchi-Levi, D. (2012). Approximating the nonlinear newsvendor and single-item stochastic lot-sizing problems when data is given by an oracle. *Operations Research, 60*, 429–446.

Hardin, J. R., Nemhauser, G. L., & Savelsbergh, M. W. P. (2007). Analysis of bounds for a capacitated single-item lot-sizing problem. *Computers and Operations Research, 34*, 1721–1743.

Hochbaum, D. S. (1995). A non-linear knapsack problem. *Operations Research Letters, 17*, 103–110.

Hochbaum, D. S. (2007). Complexity and algorithms for nonlinear optimization problems. *Annals of Operations Research, 153*, 257–296.

Hochbaum, D. S., & Hong, S. P. (1995). About strongly polynomial time algorithms for quadratic optimization over submodular constraints. *Mathematical Programming A, 69*, 269–309.

Ibarra, O. H., & Kim, C. E. (1975). Approximation algorithms for certain scheduling problems. *Mathematics of Operations Research, 4*, 197–204.

Janiak, A., Kovalyov, M. Y., Kubiak, W., & Werner, F. (2005). Positive half-products and scheduling with controllable processing times. *European Journal of Operational Research, 165*, 416–422.

Kellerer, H., Mansini, R., Pferschy, U., & Speranza, M. G. (2003). An efficient fully polynomial approximation scheme for the subset-sum problem. *Journal of Computing and System Sciences, 6*, 349–370.

Kellerer, H., & Pferschy, U. (1999). A new fully polynomial time approximation scheme for the knapsack problem. *Journal of Combinatorial Optimization, 3*, 59–71.

Kellerer, H., & Pferschy, U. (2004). Improved dynamic programming in connection with an FPTAS for the knapsack problem. *Journal of Combinatorial Optimization, 8*, 5–11.

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack problems*. Berlin: Springer.

Kellerer, H., Sarto Basso, R., & Strusevich, V. A. (2017). Approximability issues for unconstrained and constrained maximization of half-product related functions. *Theoretical Computer Science, 659*, 64–71.

Kellerer, H., & Strusevich, V. A. (2010a). Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications. *Algorithmica, 57*, 769–795.

Kellerer, H., & Strusevich, V. A. (2010b). Minimizing total weighted earliness-tardiness on a single machine around a small common due date: An FPTAS using quadratic knapsack. *International Journal of Foundations of Computer Science, 21*, 357–383.

Kellerer, H., & Strusevich, V. A. (2012). The symmetric quadratic knapsack problem: Approximation and scheduling applications. *4OR – A Quarterly Journal of Operations Research, 10*, 111–161.

Kellerer, H., & Strusevich, V. A. (2013). Fast approximation schemes for boolean programming and scheduling problems related to positive convex half-product. *European Journal of Operational Research, 228*, 24–32.

Kellerer, H., & Strusevich, V. A. (2016). Optimizing the half-product and related quadratic boolean functions: approximation and scheduling applications. *Annals of Operations Research, 240*, 39–94.

Kern, W., & Woeginger, G. J. (2007). Quadratic programming and combinatorial minimum weight product problems. *Mathematical Programming, 110*, 641–649.

Kovalyov, M. Y. (1996). A rounding technique to construct approximation algorithms for knapsack and partition-type problems. *Applied Mathematics and Computer Science, 6*(4), 789–801.

Kovalyov, M. Y., & Kubiak, W. (2012). A generic FPTAS for partition type optimisation problems. *International Journal of Planning and Scheduling, 1*, 209–233.

Lawler, E. L. (1979). Fast approximation schemes for knapsack problems. *Mathematics of Operations Research, 4*, 339–356.

Mittal, S., & Schulz, A. S. (2013). An FPTAS for optimizing a class of low-rank functions over a polytope. *Mathematical Programming, 141*, 103–120.

Qian, F., Strusevich, V. A., Gribkovskaia, I., & Halskau, Ø. (2015). Minimization of passenger takeoff and landing risk in offshore helicopter transportation: Models, approaches and analysis. *Omega, 51*, 93–106.
Rustogi, K., & Strusevich, V. A. (2013). Parallel machine scheduling: Impact of adding an extra machine. *Operations Research, 61*, 1243–1257.
Sahni, S. (1977). General techniques for combinatorial approximation. *Operations Research, 25*, 920–936.
Sarto Basso, R., & Strusevich, V. A. (2017). Differential approximation schemes for half-product related functions and their scheduling applications. *Discrete Applied Mathematics, 217*, 71–78.
Shabtay, D., Gaspar, N., & Kaspi, M. (2013). A survey on offline scheduling with rejection. *Journal of Scheduling, 16*, 3–28.
Woeginger, G. J. (2000). When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? *INFORMS Journal on Computing, 12*, 57–74.
Xu, Z. (2012). A strongly polynomial FPTAS for the symmetric quadratic knapsack problem. *European Journal of Operational Research, 218*, 377–381.