ELSEVIER

Operations Research Letters 17 (1995) 103–110

operations research letters

A nonlinear Knapsack problem

Dorit S. Hochbaum

School of Business Administration and Department of IEOR, University of California, Berkeley, CA 94720, USA

Received 1 February 1994; revised 1 May 1994

To the memory of Eugene Lawler

Abstract

The nonlinear Knapsack problem is to maximize a separable concave objective function, subject to a single "packing" constraint, in nonnegative variables. We consider this problem in integer and continuous variables, and also when the packing constraint is convex. Although the nonlinear Knapsack problem appears difficult in comparison with the linear Knapsack problem, we prove that its complexity is similar. We demonstrate for the nonlinear Knapsack problem in n integer variables and knapsack volume limit B, a fully polynomial approximation scheme with running time $\tilde{O}((1/\varepsilon^2)(n + 1/\varepsilon^2))$ (omitting polylog terms); and for the continuous case an algorithm delivering an ε-accurate solution in $O(n \log(B/\varepsilon))$ operations.

Keywords: Convex optimization; Fully polynomial approximation scheme; Knapsack problem; Quadratic Knapsack problem; Allocation problem

0. Introduction

The nonlinear Knapsack problem (NLK) is a generalization of the well-known integer Knapsack problem which maximizes a linear objective function representing utilities associated with choosing items (the number of units of item j is represented by the variable x_j) subject to a "packing" constraint:

$$\max\left\{\sum_{j=1}^{n} p_j x_j \,\middle|\, \sum_{j=1}^{n} a_j x_j \leqslant B,\ u_j \geqslant x_j \geqslant 0 \text{ and } x_j \text{ integer for } j = 1, \ldots, n\right\}.$$

In its most general form considered here the nonlinear Knapsack problem has the objective separable concave and the packing constraint separable convex:

$$\begin{aligned} \text{Max} \quad & \sum_{j=1}^{n} f_j(x_j) \\ \text{s.t.} \quad & \sum_j g_j(x_j) \leqslant B, \qquad \text{(NLK)} \\ & 0 \leqslant x_j \leqslant u_j, \quad x_j \text{ integer}, \quad j = 1, \ldots, n. \end{aligned}$$

The functions f_j are assumed concave and nondecreasing, and the functions g_j are assumed convex

SSDI 0167-6377(95)00009-7

and nondecreasing. Without loss of generality, B and u_j are integers.

There are several complexity issues that arise when the functions f_j and g_j take real values on integer inputs. In this case, the computation model assumed is the *unit cost model*, i.e. any arithmetic operation or comparison is counted as a single operation, even if the operands are real numbers. Another issue is the interpretation of what it means to solve a problem in reals. Such issues are investigated in Section 2.

Well studied classes of (NLK) are the quadratic Knapsack with the functions f_j quadratic and convex and the functions g_j linear, and the nonlinear Knapsack problem in which the functions g_j are linear. Also the continuous version of the problem is important and comes up in applications such as production planning [18], marketing [12], capital budgeting [14] and numerous others.

There is an extensive literature on solution methods for the nonlinear Knapsack problem and its special cases, and the few mentioned here form only a partial list. This literature focuses on cases where the functions are analytic and differentiable. For the integer problem, Marsten and Morin [13] developed a branch-and-bound approach combined with a dynamic programming approach developed earlier in [16]. Mathur et al. [14] also developed a branch-and-bound approach. They noted the representation of the objective function as a piecewise linear function, but as they were not aware of an efficient approach to exploit this structure they dismiss this approach as unlikely to lead to improved algorithms. This representation is in fact the key to the algorithms devised here. Using it, we establish that the continuous solution to the nonlinear Knapsack problem is solvable efficiently and in polynomial time.

Ziegler described in [18] an application with a specific objective function and packing constraint. He described for this problem a fully polynomial approximation scheme. We demonstrate that such an approximation scheme exists for all instances of (NLK), and that Ziegler's scheme results as a corrollary of the general case.

Recently, Bretthauer and Shetty [1, 2] described a branch-and-bound algorithm for the quadratic case [2], and for the nonlinear case [1]. The branch-and-bound algorithm is based on solving the continuous relaxation first. For solving the continuous problem, they rely on the availability of derivatives in analytic form, and the solution of analytic equations. Since solving the continuous relaxation is itself a challenging problem in their approach, the algorithm is only applicable to limited cases of the nonlinear Knapsack problem.

The results reported in this paper indicate that the complexity of the nonlinear Knapsack problem is quite close to that of the linear one. We consider solving optimally the continuous nonlinear Knapsack problem and present a fully polynomial approximation scheme for the (integer) nonlinear Knapsack.

We use the concept of ε-accuracy to solve the continuous problem. For a prespecified accuracy of ε, a solution is said to be ε-accurate if it is at most at a distance of ε (in the L_∞ norm) from an optimal solution. In other words, it is identical to the optimum in $O(\log 1/\varepsilon)$ significant digits. The continuous problem is shown to be solvable with an ε-accurate solution in time $O(n \log B/\varepsilon)$. This running time is not possible to improve as it is equal to the running time for solving the continuous allocation problem, which is a simple special case of the nonlinear Knapsack problem. In [6] it is proved that this running time for the allocation problem is optimal for the comparison model in that it meets the established lower bound, and cannot be substantially reduced even for the algebraic tree model that permits arithmetic operations in addition to comparisons and branchings and the floor operation.

Finally, we show an adaptation of the fully polynomial approximation scheme for the linear integer Knapsack problem, to the nonlinear case as well, with similar complexity.

We use here a terminology that is common in the context of the Knapsack problem. An item is said to be selected if the corresponding variable is set to one. The objective function coefficients are referred to as profits, and the single constraint as the packing constraint. The coefficients in the packing constraint are referred to as weights, and the value of the right-hand side of the constraint, as the Knapsack volume. We use boldface letters to denote vectors, $\|\cdot\|_\infty$ denotes the maximum norm and

e denotes the vector of all 1's $(1, \ldots, 1)$ of compatible dimension. All logarithms are base 2.

In Section 1 we describe the piecewise linear approximation that is used for the algorithms devised here. Section 2 presents a polynomial algorithm for the continuous nonlinear Knapsack problem, and in Section 3 we provide the description of the fully polynomial approximation scheme.

1. The piecewise linear approximation

A piecewise linear approximation of the functions f_j and the functions g_j is used to convert the nonlinear Knapsack problem (NLK) into a 0/1 Knapsack problem. The integer (NLK) is equivalent to the problem, (PLK), derived by a piecewise linear approximation on the integer grid. This is achieved by replacing each variable x_j by the sum of binary variables $\sum_{i=1}^{u_j} x_{ij}$, and letting $p_{ij} = f_j(i) - f_j(i-1)$, and $a_{ij} = g_j(i) - g_j(i-1)$:

$$\text{Max} \quad \sum_{j=1}^{n} \sum_{i=1}^{u_j} p_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} \sum_{i=1}^{u_j} a_{ij} x_{ij} \leqslant B, \qquad \text{(PLK)}$$

$$x_{ij} \in \{0, 1\}, \quad i = 1, \ldots, u_j, \quad j = 1, \ldots, n.$$

(PLK) is an ordinary 0/1 Knapsack problem, except that the coefficients p_{ij} or a_{ij} may not be integers or rationals. In order for this formulation to be valid for (NLK) the variables in a solution vector x_{ij} must satisfy for each given j, that when ordered as an array $(x_{ij}, \ldots, x_{u_j,j})$, two 1's may not be separated by a 0, or two 0's by a 1.

It is easy to see that the concavity of f_j and the convexity of g_j guarantee this condition. It follows that when a_{ij} and p_{ij} are integers, then techniques that are used for the 0/1 Knapsack problem are applicable here as well. There is a known dynamic programming algorithm for the 0/1 Knapsack problem:

$$\max \left\{ \sum_{j=1}^{N} p_i x_j \,\middle|\, \sum_{j=1}^{N} a_j x_j \leqslant B, x_j \geqslant 0 \text{ and } x_j \text{ integer for } j = 1, \ldots, N \right\}.$$

The complexity of solving this 0/1 Knapsack problem is $O(N \min\{B, P^*\})$ for P^* denoting the optimal solution value. For this dynamic programming to work, it is necessary that the objective function and constraint coefficients are integral. The algorithm runs in $O(NB)$ operations if only the constraint coefficients (the weights) are integral. It runs in time $O(NP^*)$ if only the objective function coefficients are integral. The $O(NP^*)$ dynamic programming algorithm is given in Section 3.1.

If the functions g_j map integers to integers or alternatively the functions f_j map integers to integers, then (NLK) is solvable in $O(B\sum_{j=1}^{n} u_j)$ steps (or $O(P^*\sum_{j=1}^{n} u_j)$ steps), which is the same complexity as the corresponding linear Knapsack problem.

2. Solving the continuous nonlinear Knapsack problem

The notion of solving a continuous problem is not well defined for nonlinear problems. The continuous problem has only been solved in special cases when the derivatives of the functions f_j and g_j exist. Even then it may be impossible to state the optimal solution accurately. The only case in which an exact and accurate solution can be obtained is the quadratic convex optimization problem over linear or convex quadratic constraints (see, for instance, [10]).

It is not surprising that difficulties are encountered in the continuous version since the solution could require infinite representation. For example, let a simple nonlinear Knapsack problem be given with $f_1(x_1) = 6x_1 - x_1^3, f_2(x_2) = 0$, the packing constraint $x_1 + x_2 \leqslant 2$, and the variables are binary. The optimal solution to this 2 variable problem is $(\sqrt{2}, 2 - \sqrt{2})$, which is irrational. The solution may even lack an algebraic representation, hence the output does not have a finite length. So solving a nonlinear problem is challenging even when the nonlinearities are as simple as polynomials. We therefore use the practical notion of *ε-accurate* solution (previously used in [7]) to represent the continuous solution to the problem. A solution, $x^{(\varepsilon)}$ is *ε-accurate* if there exists an optimal solution to the continuous problem, x^* such that $\|x^{(\varepsilon)} - x^*\|_\infty \leqslant \varepsilon$. That is, ε is the accuracy

required of the solution vector produced, and $\lceil \log 1/\varepsilon \rceil$ is the number of significant digits.

Using the notion of ε-accuracy, it is possible to solve some classes of constrained nonlinear optimization in polynomial time [7]. Hochbaum and Shanthikumar [7] obtained polynomial-time algorithms by using "proximity" results on the distance between optimal scaled solutions and the continuous optimal solution. (There is also a proximity result on the distance with respect to the integer optimal solution, which is not relevant for the nonlinear Knapsack problem.) Hochbaum, in [6], solved various allocation problems to ε-accuracy, using a different and tighter proximity theorem, applicable for allocation problems.

The *simple allocation problem* is closely related to the Knapsack problem. It differs from it in that the packing constraint is simpler – a sum of all variables, whereas for the Knapsack problem it is a *weighted* sum of the variables. The algorithms we develop make use of analysis done for the allocation problem and its continuous relaxation. The formulation of the simple allocation problem (SAP) is as follows:

$$\text{(SAP)} \quad \max \sum_{j=1}^{n} f_j(x_j)$$

$$\sum_{j=1}^{n} x_j \leqslant B,$$

$$0 \leqslant x_j \leqslant u_j \text{ integer} \quad j = 1, \ldots, n.$$

The extensive literature on the allocation problem and its extensions is surveyed in a book by Ibaraki and Katoh [9].

Frederickson and Johnson [5] (see also [6]) gave the most efficient algorithm known for the simple allocation problem. The running time of this algorithm is $O(n \log B/n)$. It was proved in [6] that this running time is optimal with respect to the comparison model and is close to the lower bound for the algebraic tree model. This running time depends linearly on the number of variables, but only logarithmically on the value of B.

The allocation problem may be viewed as a problem defined on n arrays of length up to B each: Given n arrays of nonnegative entries of function increments, $\{f_j(i) - f_j(i-1)\}_{i=1}^{u_j}$, for $j = 1, \ldots, n$, the allocation problem (SAP) is equivalent to finding the largest B entries in the arrays. One important property of the allocation problem that follows, is that a greedy algorithm, selecting one entry at a time according to largest entry value (function increment), until B entries are selected, delivers an optimal solution. Although this greedy algorithm is not polynomial, we shall make use of the implied property in the derivation of a fully polynomial approximation scheme for the integer (NLK).

Due to the concavity of the functions f_j, the chosen entries form a consecutive array of values and the last entry selected in array j corresponds to the integer value of the variable x_j. If there are less than B nonnegative values in the arrays, then the solution to the (SAP) consists of all the entries with positive value.

Hochbaum showed in [6] how to obtain an ε-accurate solution to the continuous relaxation of the allocation problem (SAP) in $O(n \log B/\varepsilon)$ steps. This solution is obtained by solving using greedy the integer problem on a scaled grid and then relying on a proximity theorem. The scaled problem is:

$$\max \sum_{j=1}^{n} f_j(s \cdot x_j)$$

$$\sum_{j=1}^{n} x_j \leqslant \frac{B}{s}$$

$$0 \leqslant x_j \leqslant \frac{u_j}{s} \text{ integer} \quad j = 1, \ldots, n.$$

Let the solution to the scaled problem be denoted by $\boldsymbol{x}^{(s)}$, and an optimal solution to the continuous problem $\boldsymbol{x}^*$. The proximity result states that for any $\boldsymbol{x}^{(s)}$ there exists an optimal continuous solution $\boldsymbol{x}^*$ so that: $\boldsymbol{x}^* \geqslant \boldsymbol{x}^{(s)} - s \cdot \boldsymbol{e}$. As a corollary, since the solution $\boldsymbol{x}^{(s)}$ satisfies $\boldsymbol{x}^{(s)} \cdot \boldsymbol{e} = B$,

$$\|\boldsymbol{x}^* - \boldsymbol{x}^{(s)}\|_\infty \leqslant n \cdot s.$$

The result is independent of the grid's uniformity.

Given the problem (PLK) we can represent it as an allocation array problem. We have n arrays, where each unit entry corresponding to the variable x_{ij} is associated with a contribution of p_{ij} units to the objective and a_{ij} units to the constraint. We replace such entry by a_{ij} entries, each with the value

p_{ij}/a_{ij} associated with it and a unit contribution to the constraint. (If $a_{ij} = 0$ then we keep only one such entry with the value p_{ij} associated with it. Such a case is unlikely, but if it occurs then the solution method is easier, as all such variables will be included.) We then solve the resulting allocation problem. This transformation corresponds to introducing new variables, $x_{ij} = y_{ij}/a_{ij}$ and solving the integer allocation problem

$$\text{Max} \quad \sum_{j=1}^{n} \sum_{i=1}^{u_j} \frac{p_{ij}}{a_{ij}} y_{ij}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} \sum_{i=1}^{u_j} y_{ij} = B, \qquad (*)$$

$$0 \leqslant y_{ij} \leqslant a_{ij},$$

$$y_{ij} \text{ integer}, \quad i = 1, \ldots, u_j \quad j = 1, \ldots, n.$$

Let the vector $\boldsymbol{y}^*$ be the optimal solution to (*). Let $\bar{y}_{ij} = (1/a_{ij}) y_{ij}^*$ for all i, j. The allocation proximity theorem implies that

$$\|\boldsymbol{x}^* - \bar{\boldsymbol{y}}\|_\infty \leqslant n \cdot \max_{ij} \left\{\frac{1}{a_{ij}}\right\}.$$

In order to obtain the ε-accuracy we modify the transformation of variables to, $x_{ij} = y_{ij}/s_{ij}$ where $s_{ij} = a_{ij}\lceil n/\varepsilon \rceil$ and duplicate each entry (i, j) s_{ij} times. Although this increases the size of the arrays, it does not cause an increase in the running time required to solved the allocation problem (*) as that depends only on the number of arrays and the right-hand side value. The right-hand side is also scaled so that all coefficients are integers:

$$\bar{B} = B\lceil n/\varepsilon \rceil.$$

Consequently, the running time is $O(n \log \bar{B}/n) = O(n \log B/\varepsilon)$.

In the quadratic Knapsack problem, the functions f_j are quadratic concave and g_j are linear. The optimal solution in this case is of polynomial length in the size of the input, so for ε polynomial in the input size an ε-accurate solution is also optimal.

The quadratic continuous Knapsack is known to be solvable in linear time [3]. The algorithm given in this section provides an alternative way for solving the continuous quadratic Knapsack. For the specified accuracy ε we duplicate each entry $1/\varepsilon a_{ij}$ times. ε is chosen so that any solution that is ε-accurate is also optimal. The resulting quadratic allocation problem is solved using the linear time algorithm in [4].

3. A fully polynomial approximation scheme

A fully polynomial approximation scheme is a family of approximation algorithms $\{A_\varepsilon\}$, where algorithm A_ε is an ε-approximation algorithm with relative error bounded by ε for all possible problem instances. In addition, the running time of A_ε depends polynomially on the input size and on $1/\varepsilon$.

A fully polynomial approximation scheme for the 0/1 Knapsack problem and the integer Knapsack problem was first devised by Ibarra and Kim [8] and later improved by Lawler [11]. Lawler further addressed the nonlinear Knapsack problem as well. The running time of his approximation scheme depends on $\sum_{j=1}^{n} u_j$. This quantity is not polynomial in the length of the input and hence the scheme is not polynomial. We demonstrate here that Lawler's fully polynomial approximation scheme for the linear Knapsack problem is implementable for (NLK) in $O((1/\varepsilon^2)(n \log B + \log(1/\varepsilon) \log n + (1/\varepsilon^2)\log(1/\varepsilon)))$.

We first present, for the sake of completeness, the simplest form of the fully polynomial approximation scheme for the 0/1 Knapsack, $\max\{\sum_{j=1}^{N} p_j x_j \mid \sum_{j=1}^{N} a_j x_j \leqslant B,\ x_j \in \{0, 1\} \text{ for } j = 1, \ldots, N\}$. Following this presentation it is demonstrated how the operations of this approximation scheme are to be modified to achieve the same result for (NLK).

3.1. An O(NP) algorithm for solving the 0/1 Knapsack problem*

The following dynamic programming algorithm solves the problem optimally for integer objective function coefficients:

Let $F_j(i)$ denote the smallest Knapsack volume that yields an objective function value of i, involving variables in the set $\{1, \ldots, j\}$. The boundary conditions are

$$F_j(0) = 0, \quad j = 1, \ldots, N$$

and the recursive formula is

$$F_j(i) = \min\{F_{j-1}(i - p_j) + a_j, F_{j-1}(i)\}.$$

Using the boundary conditions the table of values is evaluated in increasing order of objective function values:

$$F_1(1), F_2(1), \ldots, F_N(1); \quad F_1(2), \ldots, F_N(2); \ldots.$$

The computation terminates once the largest value of i is found so that $F_N(i) \leqslant B$. Each function evaluation is done in O(1) steps, and there are a total of $O(NP^*)$ function evaluations, where P^* is the optimal value of the objective function. The running time of this dynamic programming algorithm is hence $O(NP^*)$.

3.2. *A fully polynomial approximation scheme for the 0/1 Knapsack*

The idea of the fully polynomial approximation scheme is to exploit the dynamic programming algorithm that runs in $O(NP^*)$. The objective function coefficients are scaled thus reducing the running time of the algorithm to depend on the new scaled value of the optimal solution. On the other hand, for a carefully chosen scaling value the objective function of the scaled problem is close to that of the original problem.

Consider scaling the objective coefficients by a factor of k. The scaled coefficients are then $p_j(k) = \lfloor p_j/k \rfloor$, and the scaled problem, $\max\{\sum_{j=1}^{N} p_j(k)\, x_j \mid \sum_{j=1}^{N} a_j x_j \leqslant B, x_j \in \{0, 1\} \text{ for } j = 1, \ldots, N\}$. The running time required to solve the scaled problem depends on the value of the optimum. As the value of the optimal solution gets reduced by a factor of k so does the running time. We will be using upper bounds on the optimum that are also reduced by a factor of k for the scaled problem. One such simple upper and lower bound is

$$P_{\max} \leqslant P^* \leqslant N \cdot P_{\max}, \tag{3.1}$$

where $P_{\max} = \max_{j=1,\ldots,N} p_j$. With this upper bound, the running time of the dynamic programming algorithm to solve optimally the scaled problem is $O(N^2 \lfloor P_{\max}/k \rfloor)$. We now evaluate the error bound on the solution delivered by the scaled problem dynamic programming.

Let S^* be the set of indices of the variables in the optimal solution to the Knapsack problem, and $S(k)$ the set of indices of the variables in the optimal solution to the scaled problem.

$$\sum_{j \in S(k)} p_j \geqslant \sum_{j \in S(k)} k\lfloor p_j/k \rfloor \geqslant \sum_{j \in S^*} k\lfloor p_j/k \rfloor$$
$$\geqslant \sum_{j \in S^*} (p_j - k) \geqslant \sum_{j \in S^*} p_j - k|S^*|. \tag{3.2}$$

Hence, the absolute error of the "scaled" solution is at most $k\,|S^*|$, and the relative error (using the lower bound in (3.1)) is

$$\varepsilon = \frac{k|S^*|}{P_{\max}} \leqslant N\frac{k}{P_{\max}}$$

and the running time is $O(N^2 P_{\max}/k) = O(N^3 1/\varepsilon)$. Hence, this is a fully polynomial approximation scheme.

Applying this fully polynomial approximation scheme to (NLK) requires setting the objective coefficients to integers and then the running time is $O((\sum u_j)^3\, 1/\varepsilon)$ which is not polynomial. Lawler addressed the nonlinear Knapsack problem explicitly and describes an approximation scheme implemented in $O(N \log N + nN/\varepsilon)$, where $N = \sum_{j=1}^{n} u_j$ [11]. As noted before, this approximation scheme is still not polynomial due to the presence of the factor of $\sum u_j$.

In order to come up with a polynomial scheme for (NLK) some finer arguments must be used. Ibarra and Kim introduced several refinements of the algorithm involving other bounds on the optimum and the separation of items to a class of "large" ones versus "small" ones [8]. The large items are those with profits larger or equal to a given threshold value T, and the small ones have profits less than T. They also established that the error resulting from solving the large items problem followed by solving the small items problem on the remaining volume results in an error that is at most the sum of the errors.

For an improved bound consider the indices of the variables in the 0/1 Knapsack problem to be arranged in nonincreasing ratio of p_j/a_j. Let $\bar{j}$ be the largest index so that $\sum_{j=1}^{\bar{j}} a_j \leqslant B$. Let $P_0 = \max\{P_{\max}, \sum_{j=1}^{\bar{j}} p_j\}$, then $P_0 \leqslant P^* \leqslant 2P_0$. (The proof for the validity of these bounds is found in [8, 11].)

For a chosen value of ε, Lawler [11] defined large items as those with $p_j \geqslant T = \frac{1}{2}\varepsilon P_0$, and small items are all the others. The scaling factor selected is $k = \frac{1}{4}\varepsilon^2 P_0$. The reduced running time is achieved by running the dynamic programming algorithm only for the large items. The small items are then packed in order of nonincreasing p_j/a_j ratio while there is still slack in the packing constraint. The total error resulting from the union of these two sets of selected variables is at most ε. To verify that the total error does not exceed ε, note that the total error derived from the dynamic programming solution on the large items does not exceed kP^*/T as the number of large items in the optimal solution to the scaled problem is P^*/T. For the small items, the error does not exceed the value of T. The sum of the relative errors is hence

$$\frac{k}{T} + \frac{T}{P^*} \leqslant \frac{\frac{1}{4}\varepsilon^2 P_0}{\frac{1}{2}\varepsilon P_0} + \frac{\frac{1}{2}\varepsilon P_0}{P_0} = \varepsilon.$$

The computational advantage is derived from the fact that the total number of large items that needs to be considered is small. The number of different values of $p_j(k)$ is bounded by P^*/k. With the values of k and T as above, the number of different values is bounded by $8/\varepsilon^2$. As for each value of $p_j(k)$, there could be at most $n_j = \lfloor (P^*/k)/p_j(k) \rfloor$ items of size $p_j(k)$ in an optimal solution, and among all those of the same scaled profit, those with the smallest weights would be selected. Other large items need not be considered as they are dominated. With this type of argument, Lawler proved that the number of large items to be considered is $(6/\varepsilon^2)\log_2(4/\varepsilon)$. Consequently, the running time required to solve the dynamic programming algorithm that runs on large items is $O((1/\varepsilon^4)\log_2(1/\varepsilon))$.

To summarize, the steps of the approximation algorithm A_ε for the 0/1 linear Knapsack problem are as follows:

1. Find the value and the set of elements corresponding to P_0.
2. Find the "large" items that are candidates for inclusion in the optimal solution.
3. Solve for the "large" items the scaled problem, using dynamic programming.
4. Find the largest ratio "small" items that can be packed in the remaining volume of the Knapsack.

3.3. *A fully polynomial approximation scheme for the nonlinear Knapsack*

It remains to show how to implement each one of the steps of the approximation algorithm A_ε so that it applies to the nonlinear Knapsack problem in polynomial time: To find the value of P_0, we represent (NLK) as the allocation problem (*) in the variables $y_{ij} = a_{ij}x_{ij}$. Recall the greedy property of the allocation problem. For all variables y_{ij} for a fixed (i, j), the objective coefficients are the same. Hence, it is always possible to find one optimal solution in which at most one value of $x_{ij} = y_{ij}/a_{ij}$ is not an integer, but rather a fraction in the interval $(0, 1)$. From the greedy property, this variable has the next largest ratio coefficient p_{ij}/a_{ij} among all unselected items, and the smallest among all selected ones. Hence, when we round it down to 0, we get precisely the set of $\bar{j}$ items of largest ratio. The maximum of $P_{\max}$ and the objective function value corresponding to this rounded down solution, is the value of P_0.

The computationally most expensive step is step 2. There are up to $8/\varepsilon^2$ different profits in the scaled problem. For each, we need to identify the n_j ones with the least weight, and provide a pointer in each array to the index of the first such item. We first compute for each of the $8/\varepsilon^2$ arrays the position of the pointers. We proceed by computing those pointers, one at a time in decreasing value of scaled profits.

Let the current scaled profit for which we search for the pointers be q. Consider the n arrays of scaled integer profits. In each array conduct binary search to determine the first entry of value q or less. Hence, for each value of q, $q = 1, \ldots, 8/\varepsilon^2$ the running time is $O(n \log B)$ for a total of $O((1/\varepsilon^2)n \log B)$ steps. Once these pointers are identified we scan the arrays for each value of scaled profit q and record the up to n_j items with smallest weight and profit q. The amount of computation required is the total number of large items $O((1/\varepsilon^2)\log(1/\varepsilon))$ multiplied by a factor of $\log n$ to maintain a sorted array of nonincreasing weights. The running time is therefore $O((1/\varepsilon^2)\log_2(1/\varepsilon)\log n)$.

Finally, to pack the largest number of largest ratio small items, as in step 4, repeat the same procedure as for step 1, and solve the allocation

problem (*) for the small items, rounding down the one fractional variable resulting in the optimal solution. The running time of steps 1 and 4 is dominated by that of steps 2 and 3. Steps 1 and 4 require $O(n\log(B/n))$ operations each. Step 2 requires $O(n(1/\varepsilon^2)\log B + (1/\varepsilon^2)\log(1/\varepsilon)\log n)$ operations. Step 3 requires $O((1/\varepsilon^4)\ \log_2(1/\varepsilon))$ operations. The total computation time is hence $O((1/\varepsilon^2)(n\log B + \log(1/\varepsilon)\log n + (1/\varepsilon^2)\log(1/\varepsilon)))$. For ε chosen small enough this running time is the same as the time required to solve the approximation scheme for the 0/1 Knapsack problem and is hence of similar complexity.

References

[1] K. Bretthauer and B. Shetty, "The nonlinear resource allocation problem", manuscript, Texas A&M University, 1993.

[2] K. Bretthauer, B. Shetty and S. Syam, "A branch and bound algorithm for integer quadratic knapsack problems", Manuscript, Texas A&M University (1993), to appear in *ORSA J. Computing*.

[3] P. Brucker, "An O(*n*) algorithm for quadratic Knapsack problems", *Oper. Res. Lett.* **3**, 163–166 (1984).

[4] S. Cosares and D.S. Hochbaum, "A strongly polynomial algorithm for the quadratic transportation problem with fixed number of suppliers", *Math. Oper. Res.* **19**, 94–111 (1994).

[5] G.N. Frederickson and D.B. Johnson, "The complexity of selection and ranking in $X + Y$ and matrices with sorted columns", *J. Comput. Systems Sci.* **24**, 197–208 (1982).

[6] D.S. Hochbaum, "Lower and upper bounds for the allocation problem and other nonlinear optimization problems", *Math. Oper. Res.* **19**, 390–409 (1994).

[7] D.S. Hochbaum and J.G. Shanthikumar, "Convex separable optimization is not much harder than linear optimization", *J. Assoc. Comput. Mach.* **37**, 843–862 (1990).

[8] O.H. Ibarra and C.E. Kim, "Fast approximation algorithms for the Knapsack and sum of subset problems", *J. Assoc. Comput. Mach.* **22**, 463–468 (1975).

[9] T. Ibaraki and N. Katoh, *Resource Allocation Problems: Algorithmic Approaches*, MIT Press, New York, 1988.

[10] M.K. Kozlov, S.P. Tarasov and L.G. Khachian, "Polynomial solvability of convex quadratic programming", *Doklady Akad. Nauk SSR* **5**, 1051–1053 (1979). [Translated in *Sov. Math. Doklady* **20**, 1108–1111 (1979).

[11] E. Lawler, "Fast approximation algorithms for Knapsack problems", *Math. Oper. Res.* **4**, 339–356 (1979).

[12] H. Luss and S.K. Gupta, "Allocation of effort resources among competing activities", *Oper. Res.* **23**, 360–366 (1975).

[13] R.E. Marsten and T.L. Morin, "A hybrid approach to discrete mathematical programming", *Math. Programming* **14**, 21–40 (1978).

[14] K. Mathur, H.M. Salkin and S. Morito, "A Branch and search algorithm for a class of nonlinear Knapsack problems", *Oper. Res. Lett.* **2**, 155–160 (1983).

[15] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, Wiley, New York, 1990.

[16] T.L. Morin and R.E. Marsten, "An algorithm for nonlinear Knapsack problems", *Management Sci.* **22**, 1147–1158 (1976).

[17] E. Tardos, "A strongly polynomial algorithm to solve combinatorial linear programs", *Oper. Res.* **34**, 250–256 (1986).

[18] H. Ziegler, "Solving certain singly constrained convex optimization problems in production planning", *Oper. Res. Lett.* **1**, 246–252 (1982).