Solving a Class of Multiplicative Programs with the 0-1 Knapsack Constraint

Takahito Kuno*

December 5, 1997

ISE-TR-97-147

Institute of Information Sciences and Electronics
University of Tsukuba
Tsukuba, Ibaraki 305, Japan
Phone: +81-298-53-5540, Fax: +81-298-53-5206, E-mail: takahito@is.tsukuba.ac.jp

* The author was partly supported by Grant-in-Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture, Grant No. (C2)09680413.

Solving a Class of Multiplicative Programs with the 0-1 Knapsack Constraint

Takahito Kuno*
takahito@is.tsukuba.ac.jp
Institute of Information Sciences and Electronics
University of Tsukuba

December 1997

Abstract. We develop a branch-and-bound algorithm to solve a nonlinear class of 0-1 knapsack problems. The objective function is a product of $m \geq 2$ affine functions, variables of which are mutually exclusive. The branching procedure in the proposed algorithm is the usual one; but the bounding procedure exploits the special structure of the problem and implemented through two stages: the first is based on the linear programming relaxation and the second is the Lagrangian relaxation. Computational results indicate that the algorithm is promising.
Key words: Multiplicative programming, 0-1 knapsack problem, concave minimization, branch-and-bound algorithm, Lagrangian relaxation.

1. Introduction

Let us consider a nonlinear class of 0-1 knapsack problems:

$$(\mathrm{P})\quad \left| \begin{array}{ll} \text{minimize} & z = \prod_{i=1}^{m} \left(\sum_{j \in N_i} c_j x_j + d_i \right) \\ \text{subject to} & \sum_{j=1}^{n} a_j x_j \geq b \\ & x_j \in \{0,1\}, \quad j \in N = \{1, \ldots, n\}, \end{array} \right.$$

where all the data are integers and N_is are mutually exclusive, i.e.

$$N_i \cap N_h = \emptyset \text{ for } i \neq h. \tag{1.1}$$

We assume that b, c_js and d_is are positive and that

$$\sum_{j \in N} a_j \geq b. \tag{1.2}$$

Under these conditions, problem (P) is feasible; the objective function is pseudoconcave [2]; and we can assume, without loss of generality, that a_js are positive and

$$\bigcup_{i=1}^{m} N_i = N. \tag{1.3}$$

*The author was partially supported by Grant-in-Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture, Grant No. (C2)09680413.

Minimization of a product of $m \geq 2$ affine functions, so-called *multiplicative programming*, has abundant applications, including multiple objective decision making [7, 9] and geometrical optimization [12, 13]. Problem (P) can also be thought of as an m-objective optimization problem, where the costs of certain activities have no common scale if they belong to different departments. For the multiplicative program with continuous variables, a number of deterministic algorithms have been proposed so far (the readers are referred to [10, 11] for the state-of-the-art in multiplicative programming). Although the problem is NP-hard even when $m = 2$ [15], each of these algorithms is fairly efficient as long as m is, say, below five; the running time is, however, exponential in m and increases rapidly the moment m exceeds five. At this stage, there are two possible approaches to the problem with larger m: applying heuristic methods, and exploiting special structures possessed by each problem example. In their recent article [4], Benson and Boger have adopted the first approach and obtained an excellent result.

In this paper, we develop a branch-and-bound algorithm to solve (P) with every m, by exploiting (1.1) and the 0-1 knapsack constraint. The branching procedure in the proposed algorithm is the usual one, where the value of some free variable is fixed at one or zero to define a subproblem; but the bounding procedure makes the most of the structure (1.1) and is implemented through two stages: the first is based on a linear programming relaxation of the subproblem and the second is a Lagrangian relaxation. In Section 2, we will give these two relaxations in detail. In Section 3, we will show that this two-stage bounding procedure can be carried out in $O(n)$ time if the ratios c_j/a_j, $j \in N_i$, are preliminary sorted for each i. Computational results of the algorithm will be reported in Section 4.

2. Relaxations

Let us denote by (P_k) a subproblem of (P), in which some of the variables are fixed at either one or zero; and let

$$
\begin{aligned}
J_+ &= \{j \in N \mid \text{the value of } x_j \text{ is fixed at one in } (\mathrm{P}_k)\} \\
J_0 &= \{j \in N \mid \text{the value of } x_j \text{ is fixed at zero in } (\mathrm{P}_k)\} \\
F &= N \setminus (J_+ \cup J_0); \quad F_i = N_i \cap F, \quad i = 1, \ldots, m.
\end{aligned}
$$

Subproblem (P_k) is then written as follows:

$$
(\mathrm{P}_k) \quad \left| \begin{array}{ll} \text{minimize} & z = \prod_{i=1}^{m} \left(\sum_{j \in F_i} c_j x_j + d_i^k \right) \\ \text{subject to} & \sum_{j \in F} a_j x_j \geq b^k \\ & x_j \in \{0,1\}, \quad j \in F, \end{array} \right.
$$

where

$$d_i^k = d_i + \sum_{j \in J_+ \cap N_i} c_j, \quad i = 1, \ldots, m; \quad b^k = b - \sum_{j \in J_+} a_j.$$

In the sequel, we suppose that (P_k) satisfies

$$\sum_{j \in F} a_j \geq b^k > 0$$

and hence has an optimal solution $\boldsymbol{x}^k$ of value $z^k = \prod_{i=1}^m \left(\sum_{j \in F_i} c_j x_j^k + d_i^k \right)$.

Since c_js and d_is are positive, $\sum_{j \in F_i} c_j x_j + d_i$ takes a positive value at any nonnegative x_j, $j \in F_i$. This allows us to transform (P_k) into an equivalent problem:

$$(\mathrm{Q}_k) \left| \begin{array}{lll} \text{minimize} & w = \displaystyle\sum_{i=1}^m \log \left(\sum_{j \in F_i} c_j x_j + d_i^k \right) \\ \text{subject to} & \displaystyle\sum_{j \in F} a_j x_j \geq b^k \\ & x_j \in \{0, 1\}, \quad j \in F. \end{array} \right.$$

Proposition 2.1. If $(\boldsymbol{x}^k, z^k)$ is optimal to (P_k), then $(\boldsymbol{x}^k, \log z^k)$ solves (Q_k); conversely, if $(\boldsymbol{x}^k, w^k)$ is optimal to (Q_k), then $(\boldsymbol{x}^k, 2^{w^k})$ solves (P_k).

It should be noted on (Q_k) that under condition (1.1) the objective function is separable into m concave functions, each of free variables x_j, $j \in F_i$. As a solution to such separable nonconvex programs, the branch-and-bound algorithm proposed by Falk and Soland is often employed [6], Their bounding procedure uses a relaxed problem of minimizing the convex envelop of the objective function. Our first relaxation of (Q_k) is converted from Falk-Soland's for 0-1 knapsack problems.

2.1. Linear programming relaxation

For $i = 1, \ldots, m$, let us suppose that the free variables x_j, $j \in F_i$, are arranged in the increasing order of $n_i = |F_i|$ ratios:

$$c_{j_1}/a_{j_1} \leq c_{j_2}/a_{j_2} \leq \cdots \leq c_{j_{n_i}}/a_{j_{n_i}}. \tag{2.1}$$

If $F_i \neq \emptyset$, let

$$\begin{aligned} \underline{b}_i &= \max \left\{ 0, b^k - \sum_{j \in F \setminus F_i} a_j \right\}, \quad \sum_{h=1}^{p-1} a_{j_h} \leq \underline{b}_i < \sum_{h=1}^{p} a_{j_h} \\ \overline{b}_i &= \min \left\{ b^k, \sum_{j \in F_i} a_j \right\}, \qquad \sum_{h=1}^{q-1} a_{j_h} < \overline{b}_i \leq \sum_{h=1}^{q} a_{j_h}, \end{aligned}$$

where $\sum_{h=1}^0 \cdot$ is understood to be zero; and define the numbers

$$l_i = \sum_{h=1}^{p-1} c_{j_h} + d_i^k, \quad u_i = \sum_{h=1}^{q} c_{j_h} + d_i^k. \tag{2.2}$$

Lemma 2.2. *Let* $y_i^k = \sum_{j\in F_i} c_j x_j^k + d_i^k$ *for* $i = 1, \ldots, m$. *Then, for each* i *with* $F_i \neq \emptyset$,

$$l_i \leq y_i^k \leq u_i. \tag{2.3}$$

Proof: Let

$$b_i^k = \max\left\{0, b^k - \sum_{j\in F\setminus F_i} a_j x_j^k\right\}, \quad \underline{b}_i' = \sum_{h=1}^{p-1} a_{j_h}, \quad \overline{b}_i' = \sum_{h=1}^{q} a_{j_h}.$$

Then y_i^k, l_i and u_i are equal to

$$\min\left\{\sum_{j\in F_i} c_j x_j \mid \sum_{j\in F_i} a_j x_j \geq \beta,\ x_j \in \{0,1\},\ j \in F_i\right\} + d_i^k$$

if we replace β by b_i^k, $\underline{b}_i'$ and $\overline{b}_i'$, respectively. By definition, we have $\underline{b}_i' \leq \underline{b}_i \leq b_i^k \leq \overline{b}_i \leq \overline{b}_i'$, from which (2.3) follows. □

Using the bounds l_i and u_i of y_i^k, let us define

$$f_i(y_i) = \begin{cases} \log d_i^k & \text{if } F_i = \emptyset \\ \dfrac{\log(u_i/l_i)}{u_i - l_i}(y_i - l_i) + \log l_i & \text{otherwise.} \end{cases}$$

Then f_i is the convex envelop of the logarithmic function over the interval $[l_i, u_i]$ and satisfies

$$f_i(y_i) \leq \log y_i, \quad \forall y_i \in [l_i, u_i],$$

where $l_i = u_i = d_i$ if $F_i = \emptyset$. Replacing log by f_i in (Q_k) and relaxing the 0-1 variables into real ones, we have a continuous linear knapsack problem:

$$(\bar{\mathrm{Q}}_k)\ \left|\ \begin{array}{lll} \text{minimize} & w = & \sum_{j\in F} c_j^k x_j + d^k \\ \text{subject to} & & \sum_{j\in F} a_j x_j \geq b^k \\ & & 0 \leq x_j \leq 1, \quad j \in F, \end{array}\right.$$

where

$$c_j^k = \frac{\log(u_i/l_i)}{u_i - l_i} c_j, \quad j \in N_i, \quad i = 1, \ldots, m; \quad d^k = \sum_{i=1}^{m} f_i(d_i^k). \tag{2.4}$$

From the construction of ($\bar{\mathrm{Q}}_k$), we immediately see the following:

Theorem 2.3. *Let* w^k *and* $\bar{w}$ *denote the optimal values of* (Q_k) *and* $(\bar{Q}_k)$, *respectively. Then*

$$\bar{w} \leq w^k.$$

As is well known (see e.g. [5]), if we rearrange the variables x_j, $j \in F$, in the increasing order of c_j^k/a_js, then

$$\bar{w} = \sum_{h=1}^{r-1} c_{j_h}^k + \frac{c_{j_r}}{a_{j_r}} \left(b^k - \sum_{h=1}^{r-1} a_{j_h} \right),$$

and a solution $\bar{\boldsymbol{x}}$ of value $\bar{w}$ is given by

$$\bar{x}_{j_h} = \begin{cases} 1, & h = 1, \ldots, r-1 \\ (b^k - \sum_{h=1}^{r-1} a_{j_h})/a_{j_r}, & h = r \\ 0, & h = r+1, \ldots, |F|, \end{cases}$$

where

$$\sum_{h=1}^{r-1} a_{j_h} < b^k \leq \sum_{h=1}^{r} a_{j_h}. \tag{2.5}$$

We can thus use $\bar{w}$ as a lower bound of w^k to terminate branching at subproblem (Q_k) unless $\bar{w}$ is less than the incumbent value of (P). Since the time needed to solve a continuous linear knapsack problem is linear, $(\bar{\mathrm{Q}}_k)$ yielding $\bar{w}$ can also be solved in $O(n)$ time for given $[l_i, u_i]$s, without sorting c_j^k/a_js [3, 8]. Unfortunately, however, the lower bound $\bar{w}$ is not very tight as will be demonstrated in Section 4. To work the branch-and-bound algorithm efficiently on (P), we have to devise another relaxation of (Q_k) yielding a lower bound much tighter than $\bar{w}$.

2.2. Lagrangian relaxation

Let us introduce a Lagrangian multiplier $\lambda \geq 0$ into (Q_k). Then we have the second relaxation:

$$(\mathrm{L}_k(\lambda)) \left| \begin{array}{ll} \text{minimize} & w = \sum_{i=1}^{m} \log \left(\sum_{j \in F_i} c_j x_j + d_i^k \right) + \lambda \left(b^k - \sum_{j \in F} a_j x_j \right) \\ \text{subject to} & x_j \in \{0, 1\}, \quad j \in F. \end{array} \right.$$

The following is a well-known result on Lagrangian relaxation:

Lemma 2.4. Let $w(\lambda)$ denote the optimal value of $(L_k(\lambda))$. Then

$$w(\lambda) \leq w^k, \quad \forall \lambda \geq 0.$$

The question here is how we should choose a value of λ such that $w(\lambda) > \bar{w}$. To answer this, let us consider a linear programming relaxation of $(\mathrm{L}_k(\lambda))$. In the same way as we have constructed $(\bar{\mathrm{Q}}_k)$, we can linearize $(\mathrm{L}_k(\lambda))$ into

$$\left| \begin{array}{ll} \text{minimize} & w = \sum_{j \in F} (c_j^k - \lambda a_j) x_j + d^k + \lambda b^k \\ \text{subject to} & 0 \leq x_j \leq 1, \quad j \in F, \end{array} \right. \tag{2.6}$$

where c_j^ks and d^k are defined in (2.4). The optimal value $\bar{w}(\lambda)$ of (2.6) can be computed easily as follows:

$$\bar{w}(\lambda) = \sum_{j \in F} \min\{0, c_j^k - \lambda a_j\} + d^k + \lambda b^k. \tag{2.7}$$

On the other hand, the dual problem of $(\bar{\mathrm{Q}}_k)$ is of the form:

$$\left| \begin{array}{lll} \text{maximize} & w = & b^k\lambda - \sum_{j \in F} \mu_j + d^k \\ \text{subject to} & & a_j\lambda - \mu_j \le c_j^k, \quad j \in F \\ & & \lambda \ge 0, \quad \mu_j \ge 0, \quad j \in F, \end{array} \right. \tag{2.8}$$

where λ and μ_js represent the dual variables. Since (2.8) requires $\mu_j = \max\{0, a_j\lambda - c_j^k\}$ for each $j \in F$, we have

$$\bar{w} = \max_{\lambda \ge 0} \left\{ b^k\lambda - \sum_{j \in F} \max\{0, a_j\lambda - c_j^k\} \right\} + d^k. \tag{2.9}$$

Lemma 2.5. Let $(\bar{\lambda}, \bar{\boldsymbol{\mu}})$ be an optimal solution to (2.8). Then

$$\bar{w} = \bar{w}(\bar{\lambda}). \tag{2.10}$$

Proof: It follows from (2.7) and (2.9) that

$$\bar{w} = \max_{\lambda \ge 0} \bar{w}(\lambda).$$

The value $\bar{w}$ is achieved at $(\bar{\lambda}, \bar{\boldsymbol{\mu}})$ in problem (2.8); hence (2.10) follows. □

Note that the value $\bar{\lambda}$ of the dual variable is equal to the ratio $c_{j_r}^k / a_{j_r}$, where r is defined in (2.5).

Lemmas 2.4 and 2.5 imply that if we solve $(\mathrm{L}_k(\bar{\lambda}))$, we can obtain a lower bound of w^k not worse than $\bar{w}$ because (2.6) with $\lambda = \bar{\lambda}$ is a relaxed problem of $(\mathrm{L}_k(\bar{\lambda}))$.

Theorem 2.6. The relation among the optimal values of $(\bar{Q}_k)$, $(\mathrm{L}_k(\bar{\lambda}))$ and (Q_k) is

$$\bar{w} \le w(\bar{\lambda}) \le w^k, \tag{2.11}$$

where the first inequality holds strictly as long as

$$\exists i, \;\; l_i < \sum_{j \in F_i} c_j x_j(\bar{\lambda}) + d_i^k < u_i.$$

Proof: We have already shown (2.11). The latter half of the lemma follows from the strict concavity of the logarithmic function. □

2.3. Tightening the lower bound

We see from Lemma 2.2 that the optimal value of (Q_k) does not change even if we add the constraints

$$l_i \le \sum_{j\in F_i} c_j x_j + d_i^k \le u_i, \quad i = 1,\ldots,m.$$

The resulting Lagrangian relaxation with respect to $\sum_{j\in F} a_j x_j \ge b^k$ is as follows:

$$(\mathrm{L}'_k(\lambda)) \quad \left| \begin{array}{lll} \text{minimize} & w = \sum_{i=1}^{m} \log\left(\sum_{j\in F_i} c_j x_j + d_i^k\right) + \lambda\left(b^k - \sum_{j\in F} a_j x_j\right) & \\ \text{subject to} & l_i \le \sum_{j\in F_i} c_j x_j \le u_i - d_i^k, & i = 1,\ldots,m \\ & x_j \in \{0,1\}, & j \in F. \end{array} \right.$$

Since the feasible set of ($\mathrm{L}'_k(\lambda)$) is included in that of ($\mathrm{L}_k(\lambda)$), we have the following:

Theorem 2.7. Let $w'(\lambda)$ denote the optimal value of ($L'_k(\lambda)$). Then

$$\bar{w} \le w(\bar{\lambda}) \le w'(\bar{\lambda}) \le w^k.$$

While $w'(\bar{\lambda})$ is tighter than $\bar{w}$, problem ($\mathrm{L}'_k(\bar{\lambda})$) yielding the former is a 0-1 integer program in contrast to ($\bar{\mathrm{Q}}_k$). What seems to be worse, the objective function of ($\mathrm{L}'_k(\lambda)$) is nonlinear and concave. This implies that even the continuously relaxed problem of ($\mathrm{L}'_k(\lambda)$) may have multiple local minima, many of which fail to be global ones. In the next section, however, we will show that a global minimum of ($\mathrm{L}'_k(\lambda)$) can be computed in linear time if c_j/a_j, $j \in N_i$, are previously sorted for each i.

3. The algorithm

Since the sets F_i, $i = 1,\ldots,m$, of free variables are mutually exclusive, the Lagrangian relaxed problem ($\mathrm{L}'_k(\lambda)$) can be decomposed into m minimization problems, each of which is of the form:

$$\left| \begin{array}{ll} \text{minimize} & w_i = \log\left(\sum_{j\in F_i} c_j x_j + d_i^k\right) - \lambda \sum_{j\in F_i} a_j x_j \\ \text{subject to} & l_i \le \sum_{j\in F_i} c_j x_j \le u_i - d_i^k \\ & x_j \in \{0,1\}, \quad j \in F_i. \end{array} \right. \tag{3.1}$$

If we introduce an additional variable y_i, the continuously relaxed problem of (3.1) is written as follows:

$$\left| \begin{array}{ll} \text{minimize} & w_i = \log y_i - \lambda \sum_{j\in F_i} a_j x_j \\ \text{subject to} & \sum_{j\in F_i} c_j x_j + d_i^k = y_i \\ & 0 \le x_j \le 1, \quad j \in F_i; \quad l_i \le y_i \le u_i. \end{array} \right. \tag{3.2}$$

As mentioned before, this problem is neither linear nor convex; nevertheless, once the value of y_i is fixed in the interval $[l_i, u_i]$, we can solve it very easily.

Note that (3.2) with a fixed y_i is just a continuous linear knapsack problem. Therefore, the optimal value is given by

$$g_i(y_i) = \log y_i - \lambda \left(\sum_{h=1}^{s-1} a_{j_h} + \frac{a_{j_s}}{c_{j_s}} \left(y_i - \sum_{h=1}^{s-1} c_{j_h} - d_i^k \right) \right) \tag{3.3}$$

for some s such that $\sum_{h=1}^{s-1} c_{j_h} + d_i^k \leq y_i < \sum_{h=1}^{s} c_{j_h} + d_i^k$, where

$$a_{j_1}/c_{j_1} \geq a_{j_2}/c_{j_2} \geq \cdots \geq a_{j_{n_i}}/c_{j_{n_i}}. \tag{3.4}$$

Let

$$\eta_0 = d_i^k; \quad \eta_h = \eta_{h-1} + c_{j_h}, \quad h = 1, \ldots, n_i. \tag{3.5}$$

We should recall here that (3.4) is equivalent to the order (2.1) of the variables x_j, $j \in F_i$, used to compute the bounds l_i and u_i of y_i^k. Hence, from the definition (2.2), both l_i and u_i exist among η_h, $h = 1, \ldots, n_i$. We also have the following:

Lemma 3.1. The function g_i is concave on the interval $[\eta_{h-1}, \eta_h]$ for each $h = 1, \ldots, n_i$.

Proof: We see from (3.3) – (3.5) that g_i is composed of a logarithmic function and a convex piecewise affine function with break points η_h, $h = 0, 1, \ldots, n_i$. Since a sum of concave and affine functions are concave (see e.g. [14]), the function g_i is concave on each affine piece $[\eta_{h-1}, \eta_h]$. □

Lemma 3.1 guarantees that g_i is minimized at some extreme point of $[\eta_{h-1}, \eta_h]$s over the interval $[l_i, u_i]$. Moreover, if we fix the value of y_i at any $\eta_s \in \{\eta_h \mid h = 0, 1, \ldots, n_i\} \cap [l_i, u_i]$ in problem (3.2), the optimal x_{j_h} takes a 0-1 value:

$$x'_{j_h} = \begin{cases} 1 & \text{if } h \leq s \\ 0 & \text{otherwise.} \end{cases}$$

This, together with Lemma 3.1, implies that

$$g'_i = \min\{g_i(y_i) \mid y_i \in \{\eta_h \mid h = 0, 1, \ldots, n_i\} \cap [l_i, u_i]\} \tag{3.6}$$

gives the optimal value not only to (3.2) but also to the 0-1 integer program (3.1). The optimal value $w'(\lambda)$ of $(\mathrm{L}'_k(\lambda))$ can therefore be computed by

$$w'(\lambda) = \sum_{i=1}^{m} g'_i + \lambda b^k.$$

Theorem 3.2. Given $\lambda \geq 0$, problem $(L'_k(\lambda))$ can be solved in $O(n \log n)$ arithmetic operations and $O(n)$ evaluations of the logarithmic function.

Proof: For each i, sorting a_j/c_j, $j \in F_i$, in the order (3.4) requires $O(n_i \log n_i)$ arithmetic operations; and (3.6) requires $O(n_i)$ evaluations of log. Their total numbers are $O(\sum_{i=1}^{m} n_i \log n_i) = O(n \log n)$ and $O(\sum_{i=1}^{m} n_i) = O(n)$, respectively. □

This polynomial-time solvability of the nonconvex program $(\mathrm{L}'_k(\lambda))$ is totally due to the rank-two monotonicity [16, 11] possessed by the objective function of (3.1). Functions of this class are certainly concave on their domains; but the concavity can be embedded into only a two-dimensional subspace, which enable us to effectively apply parametric programming like the above (see [11] for further details).

3.1. Description of the branch-and-bound algorithm

In the preprocess of the algorithm for (P), we first sort c_j/a_j, $j \in N_i$ for each i. This requires $O(n \log n)$ arithmetic operations but omits the time needed to sort a_j/c_j, $j \in F_i$ in the solution to $(\mathrm{L}'_k(\bar{\lambda}))$ at each step after that.

procedure PREPROCESS;
begin
  for $i = 1, \ldots, m$ **do**
    sort c_j/a_j, $j \in N_i$ in the increasing order;
  set the incumbent $(\boldsymbol{x}^\circ, w^\circ) := (1, \ldots, 1, \sum_{i=1}^{m} \log(\sum_{j \in N_i} c_j + d_i))$
end;

In accordance with an ordinary branch-and-bound algorithm for 0-1 linear knapsack problems, we propose the depth-first-search rule to select (Q_k) from the set of active subproblems and as the branching variable x_t with $t = \arg\min_{j \in F} c_j^k/a_j$. Then the algorithm, incorporating the two procedures stated in Section 2, is summarized into a recursive form:

algorithm MULTI_KNAP;
begin
  PREPROCESS;
  BRANCH/BOUND$(\emptyset, \emptyset, N)$;
  $(\boldsymbol{x}^*, z^*) := (\boldsymbol{x}^\circ, 2^{w^\circ})$
end;

procedure BRANCH/BOUND(J_+, J_0, F);
begin
  let (Q_k) denote the subproblem corresponding to (J_+, J_0, F);
  if $b^k \le 0$ **then**
  begin
    for $j = 1, \ldots, n$ **do**
      if $j \in J_+$ **then** $x'_j := 1$ **else** $x'_j := 0$;

$w' := \sum_{i=1}^{m} \log(\sum_{j \in N_i} c_j x'_j + d_i)$;
if $w' < w^\circ$ **then** update the incumbent $(\boldsymbol{x}^\circ, w^\circ) := (\boldsymbol{x}', w')$
end
else if $\sum_{j \in F} a_j \geq b^k$ **then**
begin
for $i = 1, \ldots, m$ **do**
compute $[l_i, u_i]$ and define the convex envelop f_i of log over the interval;
construct the linear programming relaxation $(\bar{\mathrm{Q}}_k)$ using f_is;
solve $(\bar{\mathrm{Q}}_k)$ to obtain $\bar{w}$ and $\bar{\lambda}$;
if $\bar{w} < w^\circ$ **then**
begin
solve the Lagrangian relaxed problem $(L'_k(\bar{\lambda}))$ to obtain $w'(\bar{\lambda})$;
if $w'(\bar{\lambda}) < w^\circ$ **then**
begin
choose $t := \arg\min_{j \in F} c_j^k / a_j$, where $c_j^k = \log(u_i/l_i)\, c_j/(u_i - l_i)$;
BRANCH/BOUND($J_+ \cup \{t\}, J_0, F \setminus \{t\}$);
BRANCH/BOUND($J_+, J_0 \cup \{t\}, \setminus\{t\}$)
end
end
end
end;

Since c_j/a_j, $j \in F(i)$, have been sorted in the procedure PREPROCESS, both l_i and u_i can be computed in linear time; and hence the convex envelop f_i of log over $[l_i, u_i]$ can be obtained in linear time. This order of c_j/a_js can also be used to solve the Lagrangian relaxed problem $(\mathrm{L}'_k(\bar{\lambda}))$ and reduce the number of arithmetic operations from $O(n \log n)$ to $O(n)$. The linear programming relaxed problem $(\bar{\mathrm{Q}}_k)$ is a continuous linear knapsack problem, which can be solved in $O(n)$ time. Consequently, if an evaluation of the logarithmic function can be done in a unit time, the total computational time needed in the procedure BRANCH/BOUND is $O(n)$ before its recursive calls.

4. Computational Results

Let us report computational results of testing the algorithm MULTI_KNAP on randomly generated problems of (P).

The algorithm was coded in double precision C language (note that c_j^ks can take real values in $(\bar{\mathrm{Q}}_k)$) according to the description in the preceding section. In the procedure PREPROCESS, we sorted a_j/c_j, $j \in N_i$, by quicksort, which requires $O(n \log n)$ time on the average but $O(n^2)$ time in the worst case (see e.g. [1]). Also in the procedure BRANCH/BOUND, we solved $(\bar{\mathrm{Q}}_k)$ by sorting c_j^k/a_js with the quicksort algorithm instead of applying the linear-time algorithm to it. In addition to the code MULTI_KNAP,

Table 4.1. Comparison of MULTI_KNAP and LP_RELAX when $n = 60$.

	MULTI_KNAP						LP_RELAX	
	$\alpha = .2$		$\alpha = .5$		$\alpha = .8$		$\alpha = .2$	
m	# calls	time	# calls	time	# calls	time	# calls	time
2	40.1 (117)	.007 (.017)	88.0 (136)	.015 (.017)	175.9 (548)	.035 (.167)	6566529.6 (45904400)	775.6 (4944.1)
3	33.6 (73)	.007 (.017)	153.4 (344)	.043 (.117)	179.0 (631)	.042 (.183)		
4	35.1 (130)	.010 (.017)	282.7 (1295)	.093 (.483)	327.7 (1462)	.080 (.417)		
5	53.3 (128)	.013 (.033)	266.8 (606)	.087 (.233)	201.5 (724)	.050 (.200)	5480941.3 (32825816)	796.7 (4412.0)
6	20.6 (49)	.007 (.017)	408.0 (1591)	.137 (.600)	299.7 (1582)	.077 (.433)		
10	51.3 (172)	.015 (.050)	407.3 (960)	.143 (.383)	222.7 (683)	.065 (.250)	6174459.9 (39763636)	1174.0 (7074.6)
12	55.6 (209)	.018 (.067)	539.3 (1744)	.208 (.783)	155.3 (410)	.053 (.150)		
15	45.7 (138)	.018 (.050)	405.1 (1091)	.167 (.500)	196.9 (545)	.063 (.167)		
20	49.2 (145)	.020 (.067)	237.8 (708)	.108 (.333)	176.8 (336)	.070 (.117)	7166561.1 (53134457)	1861.2 (12909.6)
30	53.0 (118)	.025 (.083)	98.8 (182)	.052 (.100)	182.1 (357)	.092 (.183)		

we coded the algorithm omitting the second stage of bounding procedure based on the Lagrangian relaxation (denoted by LP_RELAX).

The test problems were generated in the following way: a_js and c_js were drawn from the uniform distribution in the intervals $[1, 50]$ and $[1, 20]$, respectively; b was set to the rounded value of $\alpha \sum_{j \in N} a_j$, where $0 < \alpha < 1$; and d_is were set to $d_i = \sum_{j \in N_i}(20 - a_j)$. The size of (m, n) ranged from $(2, 60)$ to $(20, 120)$; and $|N_i|$s were fixed at n/m. For each size, we solved ten instances on a UNIX workstation (hyperSPARC, 150MHz) and measured the average performance of the codes.

Table 4.1 shows the behavior of MULTI_KNAP on problems of size $n = 60$ when m increases from 2 to 30. For $\alpha = 0.2$, 0.5 and 0.8, the average number of calls on the procedure BRANCH/BOUND and the average CPU time in seconds (and their maxima in the brackets) are listed in their respective columns. It is worth noting on the results for $\alpha = 0.5$ and 0.8 that after rising the peak at some $m < 30$, the number of calls gradually decreases as m increases. The table also compares MULTI_KNAP with LP_RELAX for $\alpha = 0.2$. It clearly indicates the dominance of the Lagrangian relaxation over the linear

Table 4.2. Computational results of MULTI_KNAP when $\alpha = .5$.

m	$n = 80$ # calls	$n = 80$ time	$n = 100$ # calls	$n = 100$ time	$n = 120$ # calls	$n = 120$ time
2	113.2	.025	150.5	.045	296.4	.088
	(236)	(.067)	(269)	(.083)	(643)	(.183)
5	4387.0	2.120	3463.6	2.040	27812.4	19.795
	(37766)	(18.433)	(14031)	(8.367)	(147113)	(102.117)
10	3314.1	1.707	7096.3	4.563	99142.3	74.875
	(20559)	(10.833)	(28857)	(18.817)	(406035)	(295.050)
20	1973.8	1.312	4416.7	3.352	22389.3	20.778
	(14050)	(9.617)	(23599)	(18.783)	(134405)	(127.467)

programming relaxation.

Table 4.2 summarizes the computational results on larger-size problems. For $\alpha = 0.5$, the same statistics as in Table 4.1 are listed in it. For each size of n, the code MULTI_KNAP performs very well on problems with small m or large m (small $|N_i|$s in other words). On the whole, we can conclude that MULTI_KNAP is reasonably efficient for randomly generated problems of (P). Each test problem might be somewhat small if it were a linear 0-1 knapsack problem. However, we must not forget that the objective function of our problem (P) is nonlinear and nonconvex.

References

[1] Aho, A.V., J.E. Hopcroft and J.D. Ullman, *Data Structures and Algorithms*, Addison-Wesley (MA, 1983).

[2] Avriel, M., W.E. Diewert, S. Schaible and I. Zang, *Generalized Concavity*, Plenum Press (NY, 1988).

[3] Balas, E. and E. Zemel, "An algorithm for large zero-one knapsack problems", *Operations Research* **28** (1980) 1130 – 1154.

[4] Benson, H.P. and G.M. Boger, "Multiplicative programming problems: analysis and an efficient point search heuristic", Technical Report, College of Business Administration, University of Florida (FL, 1997), to appear in *Journal of Optimization Theory and Applications.*

[5] Dantzig, G.B., *Linear Programming and Extensions*, Princeton University Press (NJ, 1963).

[6] Falk, J.E. and R.M. Soland, "An algorithm for separable nonconvex programming problems", *Management Science* **15** (1969) 550 – 569.

[7] Geoffrion, M., "Solving bicriterion mathematical programs", *Operations Research 15* (1967) 39 – 54.

[8] Johnson, D.B. and T. Mizoguchi, "Selecting the kth element in $X + Y$ and $X_1 + X_2 + \cdots + X_m$", *SIAM Journal of Computing* **7** (1978) 147 – 153.

[9] Konno, H. and M. Inori, "Bond portfolio optimization by bilinear fractional programming", *Journal of the Operations Research Society of Japan* **32** (1988) 143 – 158.

[10] Konno, H and T. Kuno, "Multiplicative programming problems", in R. Horst and P.M. Pardalos (eds.), *Handbook of Global Optimization*, Kluwer Academic Publishers (Dortrecht, 1995).

[11] Konno, H., P.T. Thach and H. Tuy, *Global Optimization: Low Rank Nonconvex Structures*, Kluwer Academic Publishers (Dortrecht, 1997).

[12] Kuno, T., "Globally determining a minimum-area rectangle enclosing the projection of a higher-dimensional set", *Operations Research Letters* **13** (1993) 295 – 303.

[13] Maling, K., S.H. Mueller and W.R. Heller, "On finding most optimal rectangle package plans", *Proc. the 19th Design Automation Conference* (1982) 663 – 670.

[14] Mangasarian, O.L., *Nonlinear Programming*, McGraw-Hill (NY, 1969).

[15] Matsui, T., "NP-hardness of linear multiplicative programming and related problems", *Journal of Global Optimization* **9** (1996) 113 – 119.

[16] Tuy, H., "Polyhedral annexation, dualization and dimension reduction technique in global optimization", *Journal of Global Optimization* **1** (1991) 229 – 244.