A Finite Branch-and-Bound Algorithm for Linear Multiplicative Programming

Takahito Kuno*

March 9, 1999

ISE-TR-99-159

Institute of Information Sciences and Electronics
University of Tsukuba
Tsukuba, Ibaraki 305-8573, 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.

A Finite Branch-and-Bound Algorithm for Linear Multiplicative Programming

Takahito Kuno*
Institute of Information Sciences and Electronics
University of Tsukuba

March 1999

Abstract. On the basis of Soland's rectangular branch-and-bound, we develop an algorithm for minimizing a product of p (≥ 2) affine functions over a polytope. To tighten the lower bound on the value of each subproblem, we install a second-stage bounding procedure, which requires $O(p)$ additional time in each iteration but remarkably reduces the number of branching operations. Computational results indicate that the algorithm is practical if p is less than 15, both in finding an exact optimal solution and an approximate solution.

Key words: Global optimization, nonconvex optimization, linear multiplicative programming, branch-and-bound algorithm, continuous knapsack problem.

1. Introduction

In this paper, we will describe an algorithm for linear multiplicative programming, i.e., minimization of a product of p (≥ 2) affine functions over a polytope [14, 15, 17]. It is known that a product of affine functions need not be (quasi)convex [2]; and hence the problem can have multiple locally optimal solutions, many of which fail to be globally optimal. In other words, linear multiplicative programming belongs to multiextremal global optimization [13].

In the middle 60's, Swarup [27] first studied a special case of $p = 2$ in the framework of indefinite quadratic programming to find locally optimal solutions. This nonconvex program, however, had attracted little attention until the late 80's when an intensive research was undertaken because of its potential for application in various areas, including multiple criteria optimization [11], bond portfolio optimization [18], microeconomics [12], and optimal packing and layout [20]. We can now compute a globally optimal solution very efficiently if $p = 2$, using parametric simplex algorithms [14, 19, 25, 30]; in fact, the average computational time is proved to be only polynomial in the number of variables and constraints [16], though the problem is NP-hard even when $p = 2$ [23].

In contrast to the case of $p = 2$, the more general class of problems with $p \geq 3$ is rather hard to solve. There is no definite exact solution method such as the parametric

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

simplex algorithm; but several promising algorithms have been proposed so far. All of these algorithms are based on global optimization techniques: outer approximation [21], branch-and-bound [8, 24], their hybrid [4] and polyhedral annexation [29]. They are fairly efficient when p is a small number. Unfortunately, however, their running times often explode the moment that p exceeds, say, around five. Recently, to find a way out of this, two heuristic algorithms were proposed [5, 22]; but their real abilities for problems with $p > 5$ are still unknown.

The purpose of this paper is, using a rectangular branch-and-bound algorithm [13], to solve the linear multiplicative program efficiently even if p exceeds five. Taking into account cases where the feasible set has some structures such as a network flow, we develop the algorithm on the basis of the one proposed by Soland [26], since relaxed problems to be solved in it precisely inherit the structure of the original problem. To tighten the lower bound on the value of each subproblem, we install a second-stage bounding procedure, which requires $O(p)$ additional time in each iteration but remarkably accelerate the convergence of the algorithm. In Section 2, we will transform the problem into a separable concave minimization problem in order to apply the rectangular branch-and-bound algorithm. In Section 3, after explaining Soland's relaxation used in the first stage of bounding operation, we will give the detail of the second stage for tightening the lower bound. In Section 4, we will show that the second-stage bounding procedure requires $O(p)$ arithmetic operations and $O(p)$ evaluations of the logarithmic function, and then describe the whole of the algorithm. We will report the results of numerical experiments in Section 5, and give some final remarks in Section 6.

2. Reduction to a separable concave minimization

Let us consider a linear multiplicative program of minimizing a product of p (≥ 2) affine functions:

$$\left| \begin{array}{ll} \text{minimize} & z = f(\boldsymbol{x}) \equiv \prod_{i=1}^{p} (\boldsymbol{c}_i^{\mathrm{T}} \boldsymbol{x} + d_i) \\ \text{subject to} & A\boldsymbol{x} = \boldsymbol{b}, \quad \boldsymbol{x} \geq \boldsymbol{0}, \end{array} \right. \tag{2.1}$$

where $A \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, $\boldsymbol{c}_i \in \mathbb{R}^n$ and $d_i \in \mathbb{R}$ for $i = 1, \ldots, p$. We assume that the feasible set

$$X = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}\}$$

is nonempty and bounded. According to the signs of affine terms in f, we can divide X into a family of polytopes:

$$X(\mathcal{I}) = X \cap \left\{ \boldsymbol{x} \in \mathbb{R}^n \;\middle|\; \begin{array}{ll} \boldsymbol{c}_i^{\mathrm{T}} \boldsymbol{x} + d_i \leq 0 & \text{for } i \in \mathcal{I} \\ \boldsymbol{c}_i^{\mathrm{T}} \boldsymbol{x} + d_i \geq 0 & \text{for } i \notin \mathcal{I} \end{array} \right\}, \tag{2.2}$$

where $\mathcal{I} \subseteq \{1,\ldots,p\}$. As will be shown in Appendix (see also [22]), if $X(\mathcal{I})$ is nonempty for some $\mathcal{I}$ of odd cardinality, then (2.1) reduces to a number of concave maximizations over $X(\mathcal{I})$'s. This implies that we can solve such an instance using an ordinary algorithm. Throughout the paper, we impose the following assumption on problem (2.1):

Assumption 2.1. *If $\mathcal{I}$ is of odd cardinality, then $X(\mathcal{I})$ is an empty set.*

Proposition 2.1. *Under Assumption 2.1, there is a subset $\mathcal{I}$ such that $X = X(\mathcal{I})$; moreover, such an index set $\mathcal{I}$ is unique and*

$$X \subseteq \left\{ \boldsymbol{x} \in \mathbb{R}^n \;\middle|\; \begin{array}{ll} \boldsymbol{c}_i^{\mathrm{T}}\boldsymbol{x} + d_i < 0 & \textit{for } i \in \mathcal{I} \\ \boldsymbol{c}_i^{\mathrm{T}}\boldsymbol{x} + d_i > 0 & \textit{for } i \notin \mathcal{I} \end{array} \right\}.$$

Proof: See Appendix and [22]. ■

From Proposition 2.1, we can assume that (2.1) satisfies

$$\boldsymbol{c}_i^{\mathrm{T}}\boldsymbol{x} + d_i > 0 \;\text{ for any } \boldsymbol{x} \in X, \;\; i = 1,\ldots,p, \tag{2.3}$$

by reversing the signs of $\boldsymbol{c}_i$ and d_i if necessary. Under this condition, it is known that f is pseudoconcave on X (Theorem 5.15 in [2]); and hence an optimal solution $\boldsymbol{x}^*$ to (2.1) exists among vertices of the polytope X.

Let l_i and u_i be appropriate numbers satisfying

$$0 < l_i \le f_i(\boldsymbol{x}^*) \le u_i, \;\; i = 1,\ldots,p.$$

For example,

$$\left.\begin{array}{l} l_i = \min\{\boldsymbol{c}_i^{\mathrm{T}}\boldsymbol{x} \mid \boldsymbol{x} \in X\} + d_i \\ u_i = \max\{\boldsymbol{c}_i^{\mathrm{T}}\boldsymbol{x} \mid \boldsymbol{x} \in X\} + d_i \end{array}\right\} \;\; i = 1,\ldots,p.$$

Introducing a vector $\boldsymbol{\xi}$ of p auxiliary variables ξ_i, $i = 1,\ldots,p$, we have an equivalent problem to (2.1):

$$\left|\begin{array}{lll} \text{minimize} & z = \displaystyle\prod_{i=1}^{p} \xi_i & \\ \text{subject to} & \boldsymbol{x} \in X & \\ & \boldsymbol{\xi} = C\boldsymbol{x} + \boldsymbol{d}, \;\; \boldsymbol{l} \le \boldsymbol{\xi} \le \boldsymbol{u}, & \end{array}\right. \tag{2.4}$$

where $C = (\boldsymbol{c}_1,\ldots,\boldsymbol{c}_p)^{\mathrm{T}}$, $d = (d_1,\ldots,d_p)^{\mathrm{T}}$, $\boldsymbol{l} = (l_1,\ldots,l_p)^{\mathrm{T}}$ and $\boldsymbol{u} = (u_1,\ldots,u_p)^{\mathrm{T}}$. Similar transformations are found in [4, 28]. As in [5, 24], we further transform (2.4) into

$$\text{(P)} \qquad \left|\begin{array}{lll} \text{minimize} & \omega = \varphi(\boldsymbol{\xi}) \equiv \displaystyle\sum_{i=1}^{p} \log \xi_i & \\ \text{subject to} & \boldsymbol{x} \in X & \\ & \boldsymbol{\xi} = C\boldsymbol{x} + \boldsymbol{d}, \;\; \boldsymbol{l} \le \boldsymbol{\xi} \le \boldsymbol{u}. & \end{array}\right.$$

Proposition 2.2. *If (2.1) has an optimal solution $\boldsymbol{x}^*$ of value z^*, then (P) has an optimal solution $(\boldsymbol{x}^*, C\boldsymbol{x}^* + \boldsymbol{d})$ of value $\log z^*$; conversely, if (P) has an optimal solution $(\boldsymbol{x}^*, \boldsymbol{\xi}^*)$ of value ω^*, then (2.1) has an optimal solution $\boldsymbol{x}^*$ of value 2^{ω^*}.*

Proof: It follows from the monotonicity of the logarithmic function. ■

Although the objective function φ of (P) is still concave, it is separable into p functions, each of a different single variable. This enable one to solve (P) using a branch-and-bound algorithm based upon rectangular subdivisions of the set $M = \{\boldsymbol{\xi} \in \mathbb{R}^p \mid \boldsymbol{l} \leq \boldsymbol{\xi} \leq \boldsymbol{u}\}$. Let $\mathcal{M} = \{M^k \mid k = 1, \ldots, K\}$ denote a rectangular partition of M, i.e.,

$$M^k = [l_1^k, u_1^k] \times \cdots \times [l_p^k, u_p^k], \quad k = 1, \ldots, K$$
$$\bigcup_{k=1}^{K} M^k = M, \quad \mathrm{int} M^k \cap \mathrm{int} M^h = \emptyset \ \text{ if } k \neq h.$$

The class of rectangular branch-and-bound algorithms [13] systematically narrows down a rectangular region containing $\boldsymbol{\xi}^*$, by repeating three basic steps.

Branch-and-bound procedure.

Step 1. Select an appropriate M^k from $\mathcal{M}$.

Step 2. (Bounding operation) Compute a lower bound $\overline{\omega}^k$ on the minimum value of φ over $M^k \cap \{\boldsymbol{\xi} \in \mathbb{R}^p \mid \boldsymbol{\xi} = C\boldsymbol{x} + \boldsymbol{d}, \ \boldsymbol{x} \in X\}$. If $\overline{\omega}^k \geq \varphi(\boldsymbol{\xi}^\circ)$ for the best feasible solution $(\boldsymbol{x}^\circ, \boldsymbol{\xi}^\circ)$ to (P) obtained so far, exclude M^k from further consideration.

Step 3. (Branching operation) If $\overline{\omega}^k < \varphi(\boldsymbol{\xi}^\circ)$, then divide M^k into two rectangles M^{k_1} and M^{k_2}.

It is needless to say that Step 2 holds the key to efficiency of the algorithm. In the next section, we will discuss how to find a tight lower bound $\overline{w}^k$.

3. Relaxations

Taking a rectangle M^k out of a given partition $\mathcal{M} = \{M^k \mid k = 1, \ldots, K\}$, we define the associated subproblem of (P) as follows:

$$(\mathrm{P}^k) \quad \left| \begin{array}{ll} \text{minimize} & \omega = \varphi(\boldsymbol{\xi}) = \displaystyle\sum_{i=1}^{p} \log \xi_i \\ \text{subject to} & \boldsymbol{\xi} = C\boldsymbol{x} + \boldsymbol{d}, \quad \boldsymbol{x} \in X \\ & \boldsymbol{\xi} \in M^k. \end{array} \right.$$

Step 2 of the branch-and-bound procedure requires a lower bound $\overline{\omega}^k$ on the optimal value ω^k of this problem, where ω^k is understood to be $+\infty$ if (P^k) is infeasible. A typical way of computing $\overline{\omega}^k$ is to relax (P^k) by enveloping φ from below over M^k. First, we will explain two relaxation procedures of this type: one is Falk-Soland's relaxation [9] and the other is its simplified version by Soland [26]. Then we will show how to tighten the bound yielded by Soland's relaxation.

3.1. Falk-Soland's and Soland's relaxations

The convex envelope of the logarithmic function over the interval $[l_i^k, u_i^k]$ is defined by

$$\varphi_i^k(\xi_i) = \frac{\log u_i^k - \log l_i^k}{u_i^k - l_i^k}\xi_i + \frac{u_i^k \log l_i^k - l_i^k \log u_i^k}{u_i^k - l_i^k}.$$

From the concavity of log, we see that

$$\varphi_i^k(\xi_i) \leq \log \xi_i \ \text{ if } \xi_i \in [l_i^k, u_i^k]; \ \ \varphi_i^k(\xi_i) > \log \xi_i \ \text{ otherwise.} \tag{3.1}$$

Therefore, replacing $\log \xi_i$ by $\varphi_i^k(\xi_i)$ for $i = 1, \ldots, p$, we have a relaxed problem of (P^k):

$$\left|\begin{array}{ll} \text{minimize} & \omega = \varphi^k(\boldsymbol{\xi}) \equiv \sum_{i=1}^{p} \varphi_i^k(\xi_i) \\ \text{subject to} & \boldsymbol{\xi} = C\boldsymbol{x} + \boldsymbol{d}, \ \ \boldsymbol{x} \in X \\ & \boldsymbol{\xi} \in M^k. \end{array}\right. \tag{3.2}$$

This is Falk-Soland's relaxation [9]. Since (3.2) is a linear program, we can solve it by using the simplex or interior-point algorithm. If (3.2) is infeasible, we set $\overline{\omega}^k = +\infty$ in Step 2; then M^k is excluded from consideration. Otherwise, (3.1) implies that the optimal value $\overline{\omega}_0^k$ of (3.2) is less than or equal to ω^k. Hence, $\overline{\omega}_0^k$ can serve as a lower bound $\overline{\omega}^k$ in the branch-and-bound procedure.

Soland's relaxation [26] is a further relaxation of (3.2). Namely, by dropping the last set of constraints, we have Soland's relaxation of (P^k):

$$\left|\begin{array}{ll} \text{minimize} & \omega = \varphi^k(\boldsymbol{\xi}) \\ \text{subject to} & \boldsymbol{\xi} = C\boldsymbol{x} + \boldsymbol{d}, \ \ \boldsymbol{x} \in X. \end{array}\right. \tag{3.3}$$

Since the feasible set includes that of (3.2), the optimal value $\overline{\omega}_1^k$ of (3.3) satisfies

$$\overline{\omega}_1^k \leq \overline{\omega}_0^k \leq \omega^k. \tag{3.4}$$

While (3.4) implies that $\overline{\omega}_1^k$ can also serve as a lower bound $\overline{\omega}^k$, it indicates that the branching tree associated with $\overline{\omega}_1^k$ tends to grow larger than that with $\overline{\omega}_0^k$. This tendency might cause the algorithm inefficiency. Soland's relaxation, however, has some merits that can offset this weakness; i.e., (i) it precisely inherits the structure from the original problem; (ii) it is able to terminate the algorithm with a rigorous optimal solution in finite time. Let us see point (i) below; as to point (ii), we will discuss it later.

We can make point (i) clear by putting $\boldsymbol{\xi} = C\boldsymbol{x} + \boldsymbol{d}$ back into the objective function:

$$(\mathrm{Q}_1^k) \quad \left| \begin{array}{ll} \text{minimize} & \omega = \displaystyle\sum_{i=1}^{p} \frac{\log u_i^k - \log l_i^k}{u_i^k - l_i^k}(\boldsymbol{c}_i^{\mathrm{T}}\boldsymbol{x} + d_i) + d^k \\ \text{subject to} & \boldsymbol{x} \in X, \end{array} \right.$$

where

$$d^k = \sum_{i=1}^{p} \frac{u_i^k \log l_i^k - l_i^k \log u_i^k}{u_i^k - l_i^k}.$$

Since (Q_1^k) has no side constraints, we can directly apply efficient specialized algorithms to it if X has some favorable structure such as a network flow. In addition to this, all the relaxed problems to be solved in the algorithm have common constraints. Hence, an optimal solution to (Q_1^k) remains feasible for a relaxed problem next to be solved and will recover its optimality within a few simplex pivoting operations.

3.2. Tightening the bound $\overline{\omega}_1^k$

Let us proceed to a procedure for tightening the lower bound $\overline{\omega}_1^k$ yielded by Soland's relaxation, without spoiling its strong points.

Obviously, any feasible solution $(\boldsymbol{x}, \boldsymbol{\xi})$ to (P^k) satisfies

$$\sum_{i=1}^{p} \frac{\log u_i^k - \log l_i^k}{u_i^k - l_i^k}\xi_i \geq \overline{\omega}_1^k - d^k. \tag{3.5}$$

Therefore, no feasible solution is lost even if we add (3.5) to (P^k) as a constraint. Instead, we drop the first and second sets of constraints from it. Then we have an alternative relaxed problem in the p-dimensional space:

$$\left| \begin{array}{ll} \text{minimize} & \omega = \varphi(\boldsymbol{\xi}) \\ \text{subject to} & \boldsymbol{\xi} \in L^k \cap M^k, \end{array} \right. \tag{3.6}$$

where

$$L^k = \left\{ \boldsymbol{\xi} \in \mathbb{R}^p \;\middle|\; \sum_{i=1}^{p} \frac{\log u_i^k - \log l_i^k}{u_i^k - l_i^k}\xi_i \geq \overline{\omega}_1^k - d^k \right\}.$$

Proposition 3.1. *If (3.6) is infeasible, then (P^k) is infeasible. Otherwise, for any optimal solution $\boldsymbol{\xi}'$ to (3.6) we have*

$$\overline{\omega}_1^k \leq \varphi(\boldsymbol{\xi}') \leq \omega^k, \tag{3.7}$$

$$\boldsymbol{\xi}' \in \partial L^k, \tag{3.8}$$

where $\partial \cdot$ denote the set of boundary points. The first inequality of (3.7) holds strictly if $\xi_i' \in (l_i^k, u_i^k)$ for some i.

Proof: Let $S = \{(\boldsymbol{x}, \boldsymbol{\xi}) \in \mathbb{R}^n \times \mathbb{R}^p \mid \boldsymbol{\xi} \in \partial L^k\}$. Then S supports the convex set

$$W = \{(\boldsymbol{x}, \boldsymbol{\xi}) \in \mathbb{R}^n \times \mathbb{R}^p \mid \boldsymbol{\xi} = C\boldsymbol{x} + \boldsymbol{d},\ \boldsymbol{x} \in X\}.$$

If (3.6) is infeasible, then S separates $\{(\boldsymbol{x}, \boldsymbol{\xi}) \in \mathbb{R}^n \times \mathbb{R}^p \mid \boldsymbol{\xi} \in M^k\}$ from W; in other words, the feasible set of (P^k) is empty.

Now, suppose that (3.6) has an optimal solution $\boldsymbol{\xi}'$. Then we have

$$\overline{\omega}_1^k \leq \sum_{i=1}^{p} \frac{\log u_i^k - \log l_i^k}{u_i^k - l_i^k} \xi_i' + d^k, \tag{3.9}$$

where the equality holds because log is monotone increasing and $(\log u_i^k - \log l_i^k)/(u_i^k - l_i^k) > 0$ for each i; hence (3.8) follows. Also, from (3.1) we have

$$\sum_{i=1}^{p} \frac{\log u_i^k - \log l_i^k}{u_i^k - l_i^k} \xi_i' + d^k \leq \sum_{i=1}^{p} \log(\xi'). \tag{3.10}$$

The first inequality of (3.7) follows from (3.9) and (3.10). If $\xi_i' \in (l_i^k, u_i^k)$ for some i, the inequality of (3.10) holds strictly by the strict concavity of log. ■

We see from Proposition 3.1 that (3.6) provides a better lower bound than (Q_1^k) does. Unfortunately, however, (3.6) belongs to the same class as (P^k), and is too expensive to solve repeatedly in the branch-and-bound algorithm. We need to further relax (3.6) by linearizing the objective function φ.

Since any optimal solution $\boldsymbol{\xi}'$ to (3.6) satisfies (3.8), lower and upper bounds on each component ξ_i' are given by

$$s_i^k = \min\{\xi_i \mid \boldsymbol{\xi} \in \partial L^k \cap M^k\}, \tag{3.11}$$

$$t_i^k = \max\{\xi_i \mid \boldsymbol{\xi} \in \partial L^k \cap M^k\}. \tag{3.12}$$

Note that $\partial L^k \cap M^k = \partial L^k \cap ([s_1^k, t_1^k] \times \cdots \times [s_p^k, t_p^k])$, in which the set of optimal solutions to (3.6) is included. For $i = 1, \ldots, p$, let us envelop log from below over the interval $[s_i^k, t_i^k]$ and define

$$\Phi_i^k(\xi_i) = \frac{\log t_i^k - \log s_i^k}{t_i^k - s_i^k} \xi_i + \frac{t_i^k \log s_i^k - s_i^k \log t_i^k}{t_i^k - s_i^k}.$$

Then we have a linear programming relaxation of (3.6):

$$(\mathrm{Q}_2^k) \quad \left| \begin{array}{ll} \text{minimize} & \omega = \Phi^k(\boldsymbol{\xi}) \equiv \displaystyle\sum_{i=1}^{p} \Phi_i^k(\xi_i) \\ \text{subject to} & \boldsymbol{\xi} \in \partial L^k \cap M^k. \end{array} \right.$$

Proposition 3.2. *If (Q_2^k) is infeasible, then (P^k) is infeasible. Otherwise, for any optimal solution $\boldsymbol{\xi}''$ to (Q_2^k) we have*

$$\overline{\omega}_1^k \leq \Phi^k(\boldsymbol{\xi}'') \leq \omega^k, \tag{3.13}$$

where the first inequality holds strictly if $\xi_i'' \in (l_i^k, u_i^k)$ *and* $[s_i^k, t_i^k] \subset [l_i^k, u_i^k]$ *for some* i.

Proof: Let us prove the first inequality of (3.13) when $\partial L^k \cap M^k \neq \emptyset$. For any optimal solution $\boldsymbol{\xi}''$ to (Q_2^k) we have

$$\overline{\omega}_1^k = \sum_{i=1}^{p} \frac{\log u_i^k - \log l_i^k}{u_i^k - l_i^k} \xi_i'' + d^k = \sum_{i=1}^{p} \varphi_i^k(\xi_i'') \tag{3.14}$$

Since log is concave and $[s_i^k, t_i^k] \subseteq [l_i^k, u_i^k]$, we have

$$\varphi_i^k(\xi_i) \leq \Phi_i^k(\xi_i) \text{ for all } \xi_i \in [s_i^k, t_i^k]. \tag{3.15}$$

The first inequality of (3.13) follows from (3.14) and (3.15). If $\xi_i'' \in (l_i^k, u_i^k)$ and $[s_i^k, t_i^k]$ is a proper subset of $[l_i^k, u_i^k]$, the inequality of (3.15) holds strictly for $\xi_i = \xi_i''$. ■

Let $\overline{\omega}_2^k = \Phi^k(\boldsymbol{\xi}'')$ for an optimal solution $\boldsymbol{\xi}''$ to (Q_2^k) if $\partial L^k \cap M^k \neq \emptyset$; otherwise $\overline{\omega}_2^k = +\infty$. As Proposition 3.2 suggests, we can reinforce the bounding operation based on (Q_1^k) with (Q_2^k) and modify Step 2 of the branch-and-bound procedure in the following way:

Step 2-a. (First stage of bounding operation) Set $\overline{\omega}^k := \overline{\omega}_1^k$. If $\overline{\omega}^k \geq \varphi(\boldsymbol{\xi}^\circ)$ for the incumbent $(\boldsymbol{x}^\circ, \boldsymbol{\xi}^\circ)$, exclude M^k from consideration.

Step 2-b. (Second stage of bounding operation) If $\overline{\omega}^k < \varphi(\boldsymbol{\xi}^\circ)$, set $\overline{\omega}^k := \overline{\omega}_2^k$. If $\overline{\omega}^k \geq \varphi(\boldsymbol{\xi}^\circ)$, exclude M^k from consideration.

Step 2-b is aimed at revaluing the effect of constraints $\boldsymbol{\xi} \in M^k$ in (P^k) that Soland's relaxation (Q_1^k) ignores. Problem (Q_2^k) playing the central role in Step 2-b is referred to as a *cutting-plane relaxation* of (P^k), since ∂L^k works like a cutting plane in outer approximation algorithms.

4. Algorithm

Before describing the whole of the algorithm, we will show that the total computational time spent in constructing and in solving the cutting-plane relaxation (Q_2^k) is bounded by $O(p)$.

4.1. Worst-case complexity of Step 2-b

To construct (Q_2^k), we first need to compute s_i^k and t_i^k for each $i = 1, \ldots, p$. Setting $y_j = (u_j^k - \xi_j)/(u_j^k - l_j^k)$ in (3.11) and $y_j = (\xi_j - l_j)/(u_j^k - l_j^k)$ in (3.12) for $j = 1, \ldots, p$, then we have

$$\left| \begin{array}{ll} \text{minimize} & s_i = (l_j^k - u_j^k) y_i + u_i^k \\ \text{subject to} & \displaystyle\sum_{j=1}^{p} \alpha_j y_j = \sum_{j=1}^{p} \alpha_j - \beta \\ & 0 \leq y_j \leq 1, \quad j = 1, \ldots, p, \end{array} \right. \tag{4.1}$$

$$\left|\begin{array}{ll} \text{minimize} & -t_i = (l_j^k - u_j^k)y_i - l_i^k \\ \text{subject to} & \sum_{j=1}^{p} \alpha_j y_j = \beta \\ & 0 \le y_j \le 1, \quad j = 1, \ldots, p, \end{array}\right. \tag{4.2}$$

where

$$\alpha_j = \log u_j^k - \log l_j^k, \quad j = 1, \ldots, p, \quad \beta = \overline{\omega}_1^k - \sum_{j=1}^{p} \log l_j^k.$$

If $\beta < 0$ or $\sum_{j=1}^{p} \alpha_j < \beta$, we can immediately see that $\overline{\omega}_2^k = +\infty$. Let us assume that

$$0 \le \sum_{j=1}^{p} \alpha_j \le \beta. \tag{4.3}$$

Both (4.1) and (4.2) are continuous knapsack problems having a special structure, where the cost-to-volume ratio of all the items is zero except for $(l_i^k - u_i^k)/\alpha_i^k < 0$ of the ith item. Therefore, the values of s_i^k and t_i^k can be computed in a constant time as follows:

$$\begin{aligned} s_i^k &= (l_i^k - u_i^k) \min\{1, \left(\textstyle\sum_{j=1}^{p} \alpha_j - \beta\right) / \alpha_i\} + u_i^k, \\ t_i^k &= (u_i^k - l_i^k) \min\{1, \beta/\alpha_i\} + l_i^k. \end{aligned}$$

Since α_j's and β can be computed in $O(p)$ arithmetic operations and $O(p)$ evaluations of log, we can construct (Q_2^k) in $O(p)$ time if an evaluation of log can be done in a unit time.

In the same way as (4.2), we can rewrite the cutting plane relaxation (Q_2^k) as follows:

$$\left|\begin{array}{ll} \text{minimize} & \omega = \sum_{i=1}^{p} \gamma_i y_i + \delta \\ \text{subject to} & \sum_{i=1}^{p} \alpha_i y_i = \beta \\ & 0 \le y_i \le 1, \quad i = 1, \ldots, p, \end{array}\right. \tag{4.4}$$

where

$$\begin{aligned} \gamma_i &= \frac{u_i^k - l_i^k}{t_i^k - s_i^k}(\log t_i^k - \log s_i^k), \quad i = 1, \ldots, p \\ \delta &= \sum_{i=1}^{p} \frac{(t_i^k - l_i^k) \log s_i^k - (s_i^k - l_i^k) \log t_i^k}{t_i^k - s_i^k}. \end{aligned}$$

This problem (4.4) is the usual continuous knapsack problem [7]. If the items are arranged in the order

$$\gamma_{i_1}/\alpha_{i_1} \le \gamma_{i_2}/\alpha_{i_2} \le \cdots \le \gamma_{i_p}/\alpha_{i_p}, \tag{4.5}$$

an optimal solution $\overline{\boldsymbol{y}}$ is given by

$$\overline{y}_{i_j} = \begin{cases} 1, & j = 1, \ldots, q-1 \\ (\beta - \sum_{h=1}^{q-1} \alpha_{i_h})/\alpha_{j_q}, & j = q \\ 0 & j = q+1, \ldots, p, \end{cases}$$

for some q such that $\sum_{j=1}^{q-1} \alpha_{i_j} < \beta \leq \sum_{j=1}^{q} \alpha_{i_j}$ under assumption (4.3). The value of $\overline{\omega}_2^k$ is then given by

$$\overline{\omega}_2^k = \sum_{j=1}^{q-1} \gamma_{i_j} + \frac{\gamma_{i_q}}{\alpha_{i_q}} \left(\beta - \sum_{j=1}^{q-1} \alpha_{i_j} \right).$$

The ordering (4.5) requires $O(p \log p)$ time; but Balas and Zemel [3] have shown that $\overline{\boldsymbol{y}}$ can be computed in $O(p)$ without sorting γ_j/α_j's, since q can be found in linear time as a weighted median. Thus we have the following proposition, which implies that Step 2-b in the modified branch-and-bound procedure takes only $O(p)$ additional time:

Proposition 4.1. *Given $\overline{\omega}_1^k$, the cutting plane relaxation (Q_2^k) can be constructed and be solved in $O(p)$ arithmetic operations and $O(p)$ evaluations of* log.

4.2. Description of the algorithm

In Step 3, we follow the branching rule in Soland [26]. Let $\overline{\boldsymbol{x}}^k$ be an optimal basic solution to (Q_1^k) and $\overline{\boldsymbol{\xi}}^k = C\overline{\boldsymbol{x}}^k + \boldsymbol{d}$. We choose an index r such that

$$r \in \arg\max \left\{ \log \overline{\xi}_i^k - \varphi_i^k(\overline{\xi}_i^k) \;\middle|\; i = 1, \ldots, p \right\}, \tag{4.6}$$

and divide M^k into

$$M^{k_1} = [l_1^k, u_1^k] \times \cdots \times [l_{r-1}^k, u_{r-1}^k] \times [l_r^k, \overline{\xi}_r^k] \times [l_{r+1}^k, u_{r+1}^k] \times \cdots [l_1^k, u_1^k]$$
$$M^{k_2} = [l_1^k, u_1^k] \times \cdots \times [l_{r-1}^k, u_{r-1}^k] \times [\overline{\xi}_r^k, u_r^k] \times [l_{r+1}^k, u_{r+1}^k] \times \cdots [l_1^k, u_1^k].$$

Note that $\overline{\xi}_r^k$ is an interior point of the interval $[l_r^k, u_r^k]$. Otherwise, from (3.1) and (4.6) we have

$$\varphi(\boldsymbol{\xi}^\circ) \leq \varphi(\overline{\boldsymbol{\xi}}^k) \leq \varphi^k(\overline{\boldsymbol{\xi}}^k)$$

for the incumbent $\boldsymbol{\xi}^\circ$; and the set M^k must have been excluded from consideration in Step 2-a.

This rule inhibits boundless growth of the branching tree when X is a polytope. Let V denote the set of vertices of X and let $\Upsilon_i = \{\boldsymbol{c}_i^{\mathrm{T}} \boldsymbol{x} + d_i \mid \boldsymbol{x} \in V\}$ for $i = 1, \ldots, p$. Since (Q_1^k) has no side constraints, we have $\overline{\boldsymbol{x}}^k \in V$ and $\overline{\xi}_r^k \in \Upsilon_r$. Hence, every partition set M^k, generated by the rule, has its 2^p vertices in the finite set $\Upsilon = \Upsilon_1 \times \ldots \times \Upsilon_p$. Consequently, the total number of M_k's cannot exceed $|\Upsilon|$. This is the secondary strong point of Soland's relaxation that we have mentioned in Section 3.1. For further details, see [26].

We are now ready to describe the branch-and-bound algorithm incorporating the two-stage bounding operation.

algorithm BBCUT.
begin
determine the initial rectangle $M := \{\boldsymbol{\xi} \in \mathbb{R}^p \mid \boldsymbol{l} \leq \boldsymbol{\xi} \leq \boldsymbol{u}\}$ with $\boldsymbol{l} > \mathbf{0}$;
MLIST := $\{M\}$; set the incumbent value $\omega^\circ := +\infty$; $k := 1$;
while MLIST $\neq \emptyset$ **do begin**
select a rectangle as $M^k = \{\boldsymbol{\xi} \in \mathbb{R}^p \mid \boldsymbol{l}^k \leq \boldsymbol{\xi} \leq \boldsymbol{u}^k\}$ from MLIST; /* Step 1 */
MLIST := MLIST $\setminus \{M^k\}$;
for $i = 1, \ldots, p$ **do** /* Step 2-a */
determine the convex envelope φ_i^k of log over the interval $[l_i^k, u_i^k]$;
construct the Soland's relaxation (Q_1^k) using φ_i^k's;
solve (Q_1^k) to obtain an optimal basic solution $\overline{\boldsymbol{x}}^k$ and the value $\overline{\omega}_1^k$;
if $\overline{\omega}_1^k < \omega^\circ$ **then begin** /* Step 2-b */
$\overline{\boldsymbol{\xi}}^k := C\overline{\boldsymbol{x}}^k + \boldsymbol{d}$;
if $\varphi(\overline{\boldsymbol{\xi}}^k) < \omega^\circ$ **then** update $\omega^\circ := \varphi(\overline{\boldsymbol{\xi}}^k)$ and $(\boldsymbol{x}^\circ, \boldsymbol{\xi}^\circ) := (\overline{\boldsymbol{x}}^k, \overline{\boldsymbol{\xi}}^k)$;
for $i = 1, \ldots, p$ **do begin**
compute s_i^k and t_i^k by solving (4.1) and (4.2), respectively;
determine the convex envelope Φ_i^k of log over the interval $[s_i^k, t_i^k]$
end;
construct the cutting plane relaxation (Q_2^k) using $\overline{\omega}_1^k$ and Φ_i^k's;
compute the value $\overline{\omega}_2^k$ of (Q_2^k) by solving (4.4);
if $\overline{\omega}_2^k < \omega^\circ$ **then begin** /* Step 3 */
choose $r \in \arg\max\{\log \overline{\xi}_i^k - \varphi_i^k(\overline{\xi}_i^k) \mid i = 1, \ldots, p\}$;
$M^{k_1} := [l_1^k, u_1^k] \times \cdots \times [l_{r-1}^k, u_{r-1}^k] \times [l_r^k, \overline{\xi}_r^k] \times [l_{r+1}^k, u_{r+1}^k] \times \cdots [l_1^k, u_1^k]$;
$M^{k_2} := [l_1^k, u_1^k] \times \cdots \times [l_{r-1}^k, u_{r-1}^k] \times [\overline{\xi}_r^k, u_r^k] \times [l_{r+1}^k, u_{r+1}^k] \times \cdots [l_1^k, u_1^k]$;
MLIST := MLIST $\cup \{M^{k_1}, M^{k_2}\}$
end
end;
$k := k + 1$
end;
output an optimal solution $(\boldsymbol{x}^*, \boldsymbol{\xi}^*, \omega^*) := (\boldsymbol{x}^\circ, \boldsymbol{\xi}^\circ, \omega^\circ)$ (or $(\boldsymbol{x}^*, z^*) := (\boldsymbol{x}^*, 2^{\omega^\circ})$)
end;

We should note that the algorithm is finite for the above reason without considering any tolerance for the optimal value.

5. Numerical experiments

In this section, we will report computational results of testing the algorithm BBCUT on randomly generated problems. The test problems were of the form

$$
\left|\begin{array}{ll}
\text{minimize} & z = \displaystyle\prod_{i=1}^{p}\left(\sum_{j=1}^{n'} c_{ij}x_j + d\right) \\
\text{subject to} & \displaystyle\sum_{j=1}^{n'} a_{hj}x_j \le 1.0, \quad h = 1,\ldots,m' \\
& \displaystyle -\sum_{j=1}^{n'} c_{ij}x_j \le 1.0, \quad i = 1,\ldots,p \\
& x_j \ge 0.0, \quad j = 1,\ldots,n',
\end{array}\right. \tag{5.1}
$$

where a_{hj} and c_{ij} were drawn from the uniform distribution in the intervals $[0.0, 1.0]$ and $[-1.0, 1.0]$, respectively. If we introduce $m' + p$ slack variables, (5.1) reduces to problem (2.1) of size $(m, n, p) = (m' + p, n' + m' + p, p)$. Also, note that the resulting problem satisfies condition (2.3) when $d > 1.0$.

We coded the algorithm in double precision C language according to the description in Section 4.2. As to the initial rectangle M, which is not specified in the description, we determined it by setting $l_i = d - 1.0$ and $u_i = \max\{\boldsymbol{c}_i^{\mathrm{T}}\boldsymbol{x} \mid \boldsymbol{x} \in X'\} + d$ for $i = 1,\ldots,p$, where X' denotes the feasible set of (5.1). We computed the value of u_1 using the revised simplex algorithm, and those of $u_2,\ldots,u_p$ using the parametric cost simplex algorithm [6]. To select M^k from MLIST in Step 1, we tried two selection rules:

Depth first. MLIST is maintained as a *stack*. A rectangle M^k is selected and deleted from the top of MLIST; and M^{k_1} and M^{k_2} are added in this order to the top.

Best bound. MLIST is maintained as a *priority queue*. A rectangle M^k of smallest $\overline{\omega}_2^k$ is selected and deleted form MLIST.

We denote the codes adopting the depth-first and best-bound rules by BBCUT1 and BBCUT2, respectively. In Step 2-a of both codes, we solved (Q_1^k) using the parametric cost simplex algorithm, which started with the preceding solution $\overline{\boldsymbol{x}}^{k-1}$, or with the solution giving u_p when $k = 1$. To compute the value $\overline{\omega}_2^k$ of (Q_2^k) in Step 2-b, we arranged the items of (4.4) in the order (4.5) by insertion sort, instead of applying the linear time algorithm [3] to (4.4). Hence, our codes require $O(p^2)$ time to complete Step 2-b in the worst case [1]. In addition to BBCUT1 and BBCUT2, we omitted Step 2-b and wrote a code of Soland's rectangular branch-and-bound algorithm adopting the depth-first rule (denoted by SOLAND). We carried out all experiments with these codes on a Unix workstation (hyperSPARC, 150 MHz).

5.1. Computational results

Table 5.1 compares three codes on problems of size $(m', n') = (50, 50)$ and $d = 10.0$ when p ranged from 3 to 20. Each of the columns labeled BBCUT1, BBCUT2 and SOLAND contains the average number of branching operations (denoted by *branch*), the average number of simplex pivoting operations (*pivot*), and the average CPU time

Table 5.1. Comparison of three codes when $(m', n') = (50, 50)$ and $d = 10.0$.

	BBCUT1			BBCUT2			SOLAND		
p	branch	pivot	time	branch	pivot	time	branch	pivot	time
3	5.6	186.5	.398	5.6	189.2	.457	6.5	186.7	.413
5	49.3	331.3	.863	48.7	427.3	1.17	84.9	367.3	1.07
7	95.8	425.7	1.28	95.8	544.5	1.71	254.9	549.4	2.12
10	303.5	674.3	2.70	303.6	965.3	3.82	1,716	1,544	9.93
15	2,930	4,077	24.4	2,928	9,749	52.1	(68,049)	(49,640)	(436)
20	10,939	5,893	71.4	12,182	16,331	147	—	—	—

Table 5.2. Comparison of BBCUT1 and SOLAND when $(m', n', p) = (50, 50, 10)$.

	BBCUT1			SOLAND		
d	branch	pivot	time	branch	pivot	time
2.0	3,008	5,935	25.5	3,359	6,367	28.3
5.0	275.8	827.9	3.03	3,243	3,043	19.2
10.0	303.5	674.3	2.70	1,716	1,544	9.93
20.0	501.1	665.5	3.38	1,446	1,096	7.85
50.0	646.2	617.1	3.73	810.0	668.2	4.47
100.0	484.7	485.7	2.84	488.0	486.1	2.91

in seconds (*time*) taken to solve ten problems for each p. The figures in the brackets show the average of eight problems, since two test problems could not be solved within 200,000 branching operations by SOLAND; and no test problems could when $p = 20$. The codes BBCUT1 and BBCUT2 surpass SOLAND in every respect, which proves that the second-stage branching procedure (Step 2-b) worked successfully. On the other hand, the number of pivoting operations and CPU time required by BBCUT1 are rather smaller than those by BBCUT2 while there is little difference in the number of branching operations. Unlike BBCUT2, the code BBCUT1 always selects a subset of M^{k-1} as M^k unless M^{k-1} is excluded from consideration. Hence, the difference between (Q_1^{k-1}) and (Q_1^k) in BBCUT1 is usually slight compared with that in BBCUT2. This results in less pivoting operations to recover $\boldsymbol{x}^{k-1}$ in BBCUT1.

Table 5.2 shows the behavior of BBCUT1 and SOLAND on problems of size $(m', n', p) = (50, 50, 10)$ when d ranged from 2.0 to 100.0. The same statistics as in Table 5.1 are listed. In both codes, the number of branching operations decreases as d increases, since the gap between log and its lower envelope over the interval $[l_i, u_i] = [d - 1.0, \max\{\boldsymbol{c}_i^{\mathrm{T}}\boldsymbol{x} \mid$

Table 5.3. Computational results of BBCUT1 when $d = 10.0$.

	$p = 5$			$p = 10$			$p = 15$		
$m' \times n'$	branch	pivot	time	branch	pivot	time	branch	pivot	time
50× 50	49.3	331.3	.863	303.5	674.3	2.70	2,930	4,077	24.4
50×100	27.1	453.8	1.44	463.9	1,467	7.16	4,258	6,556	50.8
100×100	31.7	593.3	4.26	704.8	2,692	26.1	5,217	9,591	130
100×150	34.3	829.9	8.86	550.6	2,806	39.0	6,084	12,385	250
150×150	39.4	965.6	20.4	674.7	3,515	91.3	6,712	18,437	605
150×200	41.4	1,334	36.3	945.8	6,807	224	8,835	32,958	1,311

Table 5.4. Performance of BBCUT2 in heuristics when $d = 10.0$.

	$p = 5$			$p = 10$			$p = 15$		
	$R \times 10^5$			$R \times 10^5$			$R \times 10^5$		
$m' \times n'$	av.	s.d.	time	av.	s.d.	time	av.	s.d.	time
50× 50	1.7	3.4	.627	0.0	0.0	1.19	0.2	0.5	1.97
50×100	0.0	0.0	1.36	4.5	11.8	2.75	0.0	0.0	4.62
100×100	0.0	0.0	4.04	1.1	2.5	8.01	1.8	5.2	12.5
100×150	0.1	0.1	8.15	0.1	0.2	17.0	0.1	0.2	27.2
150×150	0.1	0.3	18.4	1.4	2.6	38.1	0.0	0.1	57.0
150×200	0.4	1.3	31.9	0.2	0.3	63.9	1.1	2.3	94.2

$\boldsymbol{x} \in X'\} + d]$ becomes smaller for each i. At $d = 100.0$, these two codes almost coincide in the numbers of branching, pivoting operations and in the CPU time. However, it is worth noting that BBCUT1 requires only one-tenth to one-third of branching operations required by SOLAND when d is between 5.0 and 20.0; and besides its CPU time is quite stable for $d \geq 5.0$.

Table 5.3 summarizes the computational results of BBCUT1 on larger-size problems with $d = 10.0$. It contains the same statistics as before. We see from this table that, at least for the class (5.1) of randomly generated problems, the code BBCUT1 keeps its efficiency up to considerably large size (m', n') as long as p is less than 15.

5.2. Performance of BBCUT2 in heuristics

As we have seen in Table 5.1, the code BBCUT2 adopting the best-bound rule is less attractive than BBCUT1 for finding an exact solution. However, it may be of use in heuristics because BBCUT2 picks up a most hopeful rectangle M^k from MLIST in every iteration. Even if we stop the code before MLIST $= \emptyset$, it would provide a reasonably good feasible solution $\boldsymbol{x}^\circ$, the incumbent at the time. To confirm this, we stopped

BBCUT2 after $2p$ branching operations and evaluated the quality of the approximate solution $\boldsymbol{x}^\circ$ relative to a globally optimal solution $\boldsymbol{x}^*$. We employed as the quality rating of $\boldsymbol{x}^\circ$ a relative error:

$$R = \frac{f(\boldsymbol{x}^\circ) - z^*}{z^*},$$

where z^* is the optimal value of (5.1) computed by BBCUT1.

Table 5.4 shows the results on problems of the same sizes as in Table 5.3. It contains the average relative error ($R \times 10^5$, *av.*), its standard deviation ($R \times 10^5$, *s.d.*), and the average CPU time in seconds (*time*) of ten problems for each (m', n', p). We can observe in this table that BBCUT2 generates fine approximate solutions; the average relative error is less than 10^{-4} for every size. This can rank with the results of heuristic algorithms recently proposed by Benson and Boger [5] and Liu et.al. [22].

6. Concluding remarks

We have seen that BBCUT can serve as a practical algorithm both in finding an exact optimal solution and an approximate solution to the linear multiplicative program (2.1) with p less than 15. The algorithm BBCUT is on the basis of Soland's rectangular branch-and-bound algorithm and equipped with a newly designed second-stage bounding procedure. This procedure revalues the constraints for auxiliary variables $\boldsymbol{\xi} \in M^k$ that Soland's relaxation ignores. Practically, it solves a continuous knapsack problem with p variables and hence requires $O(p)$ additional time; but the algorithm can reduce the number of branching operations considerably with the help of this rather simple procedure. Since we have not compared BBCUT with other recent promising algorithms, we can make no final conclusions about its computational properties. However, the algorithm BBCUT will be effective for problems with structured constraints, especially for network flow problems, because it uses no relaxed problems with side constraints.

Before closing the paper, we touch upon an extension of the cutting plane relaxation (Q_2^k). Recall that, to show the validity of (Q_2^k), we need only the strict concavity and monotonicity of the logarithmic function. This implies that the cutting plane relaxation is applicable to other separable concave minimization problems if the objective function is strictly concave and monotonic. Such problems will abound in the class of minimum concave-cost network flows [10].

Appendix

The following are discussed in Liu et al. [22]. However, we provide them here for the benefit of the readers.

As we have seen in Section 2, the linear multiplicative program (2.1) can reduce to a family of problems:

$$(\mathrm{P}_{\mathcal{I}}) \quad \left| \begin{array}{ll} \text{minimize} & z = f(\boldsymbol{x}) \equiv \prod_{i=1}^{p} (\boldsymbol{c}_i^{\mathrm{T}} \boldsymbol{x} + d_i) \\ \text{subject to} & \boldsymbol{x} \in X(\mathcal{I}), \end{array} \right.$$

where $X(\mathcal{I})$ is defined in (2.2). By definition, we have $f(\boldsymbol{x}) \leq 0$ for all $\boldsymbol{x} \in X(\mathcal{I})$ if $|\mathcal{I}|$ is an odd number, while $f(\boldsymbol{x}) \geq 0$ for all $\boldsymbol{x} \in X(\mathcal{I})$ if $|\mathcal{I}|$ is an even number. This implies that, if $X(\mathcal{I}) \neq \emptyset$ for some $\mathcal{I}$ of odd cardinality, we need only to solve $(\mathrm{P}_{\mathcal{I}})$ for each $\mathcal{I}$ of odd cardinality to find an optimal solution to (2.1). Note that $(\mathrm{P}_{\mathcal{I}})$ with odd $\mathcal{I}$ is equivalent to a concave maximization problem:

$$\left| \begin{array}{ll} \text{maximize} & \sum_{i \in \mathcal{I}} \log(-\boldsymbol{c}_i^{\mathrm{T}} \boldsymbol{x} - d_i) + \sum_{i \notin \mathcal{I}} \log(\boldsymbol{c}_i^{\mathrm{T}} \boldsymbol{x} + d_i) \\ \text{subject to} & \boldsymbol{x} \in X(\mathcal{I}). \end{array} \right.$$

Hence, we can solve (2.1) using any one of convex minimization algorithms when it does not satisfy Assumption 2.1.

Proof of Proposition 2.1: Suppose that $X(\mathcal{I}')$ is nonempty. Then $|\mathcal{I}'|$ is an even number under Assumption 2.1. Let us take an arbitrary point $\boldsymbol{x}'$ out of $X(\mathcal{I}')$. If $\boldsymbol{c}_i^{\mathrm{T}} \boldsymbol{x}' + d_i = 0$ for some $i \in \mathcal{I}'$, then $\boldsymbol{x}' \in X(\mathcal{I}' \setminus \{i\})$. This implies that $X(\mathcal{I}' \setminus \{i\}) \neq \emptyset$ and contradicts Assumption 2.1. If $\boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}' + d_i = 0$ for some $i \neq \mathcal{I}'$, then $\boldsymbol{x}' \in X(\mathcal{I}' \cup \{i\})$, which is again a contradiction. Therefore, for any $\boldsymbol{x} \in D(\mathcal{I}')$ we have

$$\boldsymbol{c}_i^{\mathrm{T}} \boldsymbol{x} + d_i < 0 \ \text{ if } i \in \mathcal{I}'; \ \ \boldsymbol{c}_i^{\mathrm{T}} \boldsymbol{x} + d_i > 0 \ \text{ otherwise.} \tag{A.1}$$

Next, to prove the uniqueness of $\mathcal{I}'$, assume that there is an index set $\mathcal{I}'' \neq \mathcal{I}'$ such that $D(\mathcal{I}'') \neq \emptyset$. Choose an arbitrary point $\boldsymbol{x}'' \in X(\mathcal{I}'')$. Then the line segment $\boldsymbol{x}'$–$\boldsymbol{x}''$ contains a point $\boldsymbol{x} \in D(\mathcal{I}')$ such that $\boldsymbol{c}_i^{\mathrm{T}} \boldsymbol{x} + d_i = 0$ for some $i \in (\mathcal{I}' \setminus \mathcal{I}'') \cup (\mathcal{I}'' \setminus \mathcal{I}')$. This contradicts (A.1); hence only $D(\mathcal{I}')$ must be nonempty. ■

References

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

[2] Avriel, M., W.E. Diewert, S. Schaible and I. Zang, *Generalized Convexity*, Plenum Press (N.Y., 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., "An outcome space branch and bound-outer approximation algorithm for convex multiplicative programming", Technical Report, Department of Decision and Information Sciences, University of Florida (Fl., 1998).

[5] Benson, H.P. and G.M. Boger, "Multiplicative programming problems: analysis and efficient point search heuristic", *Journal of Optimization Theory and Applications* **94** (1997), 487 - 510.

[6] Chvătal, V., *Linear Programming*, W.H. Freeman (N.Y., 1983).

[7] Dantzig, G.B., "Discrete variable extremum problems", *Operations Research* **5** (1957), 266 – 277.

[8] Falk, J.E. and S.W. Palocsay, "Image space analysis of generalized fractional programs", *Journal of Global Optimization* **4** (1994), 63 – 88.

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

[10] Guisewite, G.M. and P.M. Pardalos, "Minimum concave-cost network flow problems: applications, complexity and algorithms", *Annals of Operations Research* **25** (1990), 75 – 100.

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

[12] Henderson, J.M. and R.E. Quandt, *Microeconomic Theory*, McGraw-Hill (N.Y., 1971).

[13] Horst, R. and H. Tuy, *Global Optimization: Deterministic Approaches*, 2nd ed. Springer-Verlag (Berlin, 1993).

[14] Konno, H. and T. Kuno, "Linear multiplicative programming", *Mathematical Programming* **56** (1992), 51 – 64.

[15] Konno, H. and T. Kuno, "Multiplicative programming problems", in R. Horst and P.M. Pardalos (eds.), *Handbook of Global Optimization*, Kluwer Academic Publishers (Dordrecht, 1995), 369 – 405.

[16] Konno, H., T. Kuno and Y. Yajima, "Parametric simplex algorithms for a class of NP-complete problems whose average number of steps is polynomial", *Computational optimization and applications* **1** (1992), 227 – 239.

[17] Konno, H., P.T. Thach and H. Tuy, *Optimization on Low Rank Nonconvex Structures*, Kluwer Academic Publishers (Dordrecht, 1997).

[18] Konno, H. and H. Watanabe, "Bond portfolio optimization problems and their applications to index tracking: a partial optimization approach", *Journal of Operations Research Society of Japan* **39** (1996), 295 – 306.

[19] Konno, H., Y. Yajima and T. Matsui, "Parametric simplex algorithms for solving a special class of nonconvex minimization problems", *Journal of Global Optimization* **1** (1991), 65 – 81.

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

[21] Kuno, T., Y. Yajima and H. Konno, "An outer approximation method for minimizing the product of several convex functions on a convex set", *Journal of Global Optimization* **3** (1993), 325 – 335.

[22] Liu, X.J., T. Umegaki and Y. Yamamoto, "Heuristic methods for linear multiplicative programming", Discussion Paper No.776, Institute of Policy and Planning

Sciences, University of Tsukuba (Ibaraki, 1998).

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

[24] Ryoo, H.S. and N.V. Sahinidis, "A branch-and-reduce approach to global optimization", *Journal of Global Optimization* **8** (1996), 107 – 138.

[25] Schaible, S. and C. Sodini, "Finite algorithm for generalized linear multiplicative programming", *Journal of Optimization Theory and Applications* **87** (1995), 441 – 455.

[26] Soland, R.M., "Optimal facility location with concave costs", *Operations Research* **22** (1974), 373 – 382.

[27] Swarup, K., "Programming with indefinite quadratic function with linear constraints", *Cahiers du d'Études de Recherche Opérationnelle* **8** (1966), 132 – 136.

[28] Thoai, N.V., "A global optimization approach for solving the convex multiplicative programming problem," *Journal of Global Optimization* **1** (1991), 341 – 357.

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

[30] Tuy, H. and B.T. Tam, "An efficient solution method for rank two quasiconcave minimization problems", *Optimization* **24** (1992), 43 – 56.