Bull. Malays. Math. Sci. Soc. (2024) 47:137
https://doi.org/10.1007/s40840-024-01730-3

BULLETIN of the
MALAYSIAN MATHEMATICAL
SCIENCES SOCIETY
http://www.springer.com/mathematics/journal/40840

A Self-Adjustable Branch-and-Bound Algorithm for Solving Linear Multiplicative Programming

Yanzhen Zhang[1]

Received: 23 September 2023 / Revised: 15 April 2024 / Accepted: 9 June 2024 /
Published online: 3 July 2024

Abstract
This article presents a self-adjustable branch-and-bound algorithm for globally solving a class of linear multiplicative programming problems (LMP). In this algorithm, a self-adjustable branching rule is introduced and it can continuously update the upper bound for the optimal value of LMP by selecting suitable branching point under certain conditions, which differs from the standard bisection rule. The proposed algorithm further integrates the linear relaxation program and the self-adjustable branching rule. The dependability and robustness of the proposed algorithm are demonstrated by establishing the global convergence. Furthermore, the computational complexity of the proposed algorithm is estimated. Finally, numerical results validate the effectiveness of the self-adjustable branching rule and demonstrate the feasibility of the proposed algorithm.

Keywords Global optimization · Linear multiplicative program · Linear relaxation · Branch-and-bound

Mathematics Subject Classification 90C30 · 90C26

1 Introduction

Consider the following linear multiplicative problem:

$$\text{(LMP)}: \begin{cases} \max\ f(\boldsymbol{y}) = \sum_{i=1}^{p} (\boldsymbol{c}_i^T \boldsymbol{y} + c_{0i})(\boldsymbol{d}_i^T \boldsymbol{y} + d_{0i}) \\ \text{s.t.}\quad \boldsymbol{y} \in \mathbb{Y} = \{\boldsymbol{y} \in \mathbb{R}^n \mid \boldsymbol{A}\boldsymbol{y} \le \boldsymbol{b}\}, \end{cases}$$

Communicated by Anton Abdulbasah Kamil.

✉ Yanzhen Zhang
zhangyanzhen0724@163.com

1 College of Mathematics and Information Science, Henan Normal University, Xinxiang 453007, China

where $\boldsymbol{c}_i, \boldsymbol{d}_i \in \mathbb{R}^n$, $c_{0i}, d_{0i} \in \mathbb{R}$, $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, and $\mathbb{Y} \in \mathbb{R}^n$ is a nonempty and bounded set.

In recent decades, LMP, being a specialized multiplicative program, has garnered significant attention from researchers. The prominence of LMP stems from its wide-ranging applications in financial optimization [1–3], microeconomics [4], plant layout design [5], data mining/patten recognition [6], VLISI chip design [7], robust optimization [8], and various special linear or nonlinear programming problems [9–15] that can be converted to the form of LMP. Another key aspect is that LMP is known to be NP-hard [16] and typically exhibits multiple locally optimal solutions which are not global optimal solutions of LMP [17].

Various practical algorithms have been proposed for solving LMP and its special cases. Based on their distinctive characteristics, these algorithms can be classified into branch and bound algorithms [18–21], optimal level solution methods [22, 23], monotonic optimization approaches [24], outcome-space approaches [25, 26], approximate algorithms [27], level set algorithms [28], etc. Specially, the method that introducing other technique into branch-and-bound algorithm has gained respect. For instance, Wang et al. [29, 30] presented a practicable branch-and-bound method and a novel convex relaxation-strategy-based algorithm by using a new linear relaxation technique and a convex approximation approach, respectively. Zhao et al. [31] developed a simple global optimization algorithm based on a new convex relaxation method, the branch and bound scheme and some accelerating techniques. Yin et al. [32] proposed a new outer space rectangle branch and bound algorithm by employing affine approximations technique to subsequently refine the initial outer space rectangle. Shen et al. [33–37] designed a series of global algorithms after reconsideration of the linear relaxation technique, second order cone relaxation and the convex quadratic relaxation, respectively. However, the above branch-and-bound algorithms are usually combined with the standard bisection branching rule which uses the midpoint of the longest edge of the rectangle as the branching point. So the optimal solution of divided rectangle may still be the optimal solution of sub-rectangle, i.e., the upper bound for the optimal solution of LMP over sub-rectangle is not updated during the execution of the algorithm.

Hence, a self-adjustable branch-and-bound (SABB) algorithm is presented to solve LMP. Compared with the existing literatures [29–35, 38], the features of SABB algorithm are summarized as follows: (i) In spite of the linear relaxation program being similar to the existing methods in [29, 32, 39], the objective function in our algorithm is processed prior to the implementation of the equivalent transformation, which is different from the algorithms in [29, 32]. Furthermore, this algorithm necessitates fewer auxiliary variables for solving LMP compared to those used in previous studies [29, 32]. (ii) The introduction of the self-adjustable branching rule in the proposed algorithm enables constant update to the upper bound for the optimal value of LMP by selecting an appropriate branching point in the direction of better relaxation approximation of the rectangle. And numerical results validate the effectiveness of the self-adjustable branching rule by comparing it with the standard bisection branching rule. (iii) The branching operation of the proposed algorithm is performed in the image space $\mathbb{R}^p$ rather than decision variable space $\mathbb{R}^n$ in [29, 30, 34]. This means that the proposed algorithm will help economize on the required computation if $p \ll n$.

(iv) The global convergence and computation complexity of the proposed algorithm are analyzed. However, the complexity of the algorithms in [29, 32, 34, 38] is not specified.

The rest part of this paper is organized as follows. In Sect. 2, we describe the equivalent transformation for LMP and establish its linear relaxation program. A self-adjustable branching rule is introduced in Sect. 3. In Sect. 4, we summarize the SABB algorithm and establish its the global convergence, and its complexity is estimated as well. We test some numerical examples and report numerical results in Sect. 5. Finally, some conclusions are given in Sect. 6.

2 Equivalent Problem and Its Linear Relaxation Programming

To globally solve LMP, an equivalent problem (EP) is established by an equivalent transformation. Subsequently, we propose a linear relaxation programming approach for EP, which provides an upper bound to the optimal value of LMP.

In regard to the objective function $f(\boldsymbol{y})$ of LMP, we convert it into the following form:

$$\begin{aligned} f(\boldsymbol{y}) &= \sum_{i=1}^{p}\left(\boldsymbol{c}_i^T\boldsymbol{y}+c_{0i}\right)\left(\boldsymbol{d}_i^T\boldsymbol{y}+d_{0i}\right) \\ &= \sum_{i=1}^{p}\left[\left(\boldsymbol{c}_i^T\boldsymbol{y}\right)\left(\boldsymbol{d}_i^T\boldsymbol{y}\right)+\left(d_{0i}\boldsymbol{c}_i+c_{0i}\boldsymbol{d}_i\right)^T\boldsymbol{y}+c_{0i}d_{0i}\right]. \end{aligned}$$

By introducing $2p$ auxiliary variables, we covert LMP into its EP. Let

$$z_i=\boldsymbol{c}_i^T\boldsymbol{y},\ h_i=\boldsymbol{d}_i^T\boldsymbol{y}z_i, i=1,\ldots,p,$$

and define an initial rectangle Z^0, given by

$$Z^0=\left\{\boldsymbol{z}\in\mathbb{R}^p\ |l_i^0\le z_i\le u_i^0, i=1,\ldots,p\right\},$$

where ${l_i}^0=\min_{\boldsymbol{y}\in\mathbb{Y}}\boldsymbol{c}_i^T\boldsymbol{y}$, ${u_i}^0=\max_{\boldsymbol{y}\in\mathbb{Y}}\boldsymbol{c}_i^T\boldsymbol{y}, i=1,\ldots,p$. Meanwhile, we also calculate $L_i=\min_{\boldsymbol{y}\in\mathbb{Y}}\boldsymbol{d}_i^T\boldsymbol{y}$, $U_i=\max_{\boldsymbol{y}\in\mathbb{Y}}\boldsymbol{d}_i^T\boldsymbol{y}, i=1,\ldots,p$.

Hence, LMP can be reformulated as EP over Z^0 as follows:

$$(\mathrm{EP}(Z^0)):\begin{cases}\max\ \phi(\boldsymbol{y},\boldsymbol{z},\boldsymbol{h})=\sum\limits_{i=1}^{p}[h_i+(d_{0i}\boldsymbol{c}_i+c_{0i}\boldsymbol{d}_i)^T\boldsymbol{y}+c_{0i}d_{0i}]\\ \text{s.t.}\quad h_i=\boldsymbol{d}_i^T\boldsymbol{y}z_i, i=1,\ldots,p,\\ z_i=\boldsymbol{c}_i^T\boldsymbol{y}, i=1,\ldots,p,\\ \boldsymbol{z}\in Z^0,\\ \boldsymbol{d}_i^T\boldsymbol{y}\in[L_i,U_i], i=1,\ldots,p,\\ \boldsymbol{y}\in\mathbb{Y}.\end{cases}$$

The following theorem establishes the equivalence between LMP and EP.

Theorem 1 *$\boldsymbol{y}^*$ is an optimal solution to LMP if and only if $(\boldsymbol{y}^*, \boldsymbol{z}^*, \boldsymbol{h}^*)$ is an optimal solution to EP, in which $zi^* = c_i^T x^*$, $hi = \boldsymbol{d}_i^T \boldsymbol{y}^* z_i^*$, $i = 1, 2, \ldots, p$.*

Proof This proof bears resemblance to Theorem 1 in Shen et al. [33], so it is omitted from this paper. □

According to Theorem 1, LMP and EP share the same global optimal value. Subsequently, our focus will be on addressing the solution to EP.

For convenience, assume that

$$Z = \{\boldsymbol{z} \in \mathbb{R}^p \mid l_i \leq z_i \leq u_i, i = 1, \ldots, p\} \subseteq Z^0. \tag{2.1}$$

Thus, for any $Z \subseteq Z^0$, the corresponding EP over Z can be rewritten as follows:

$$(\text{EP(Z)}) : \begin{cases} \max \phi(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{h}) = \sum_{i=1}^{p} [h_i + (d_{0i}\boldsymbol{c}_i + c_{0i}\boldsymbol{d}_i)^T \boldsymbol{y} + c_{0i}d_{0i}] \\ \text{s.t.} \quad h_i = \boldsymbol{d}_i^T \boldsymbol{y} z_i, i = 1, \ldots, p, \\ z_i = \boldsymbol{c}_i^T \boldsymbol{y}, i = 1, \ldots, p, \\ \boldsymbol{z} \in Z, \\ \boldsymbol{d}_i^T \boldsymbol{y} \in [L_i, U_i], i = 1, \ldots, p, \\ \boldsymbol{y} \in \mathbb{Y}. \end{cases}$$

From the structure of EP(Z), its non-convexity is manifested in constraints $h_i = \boldsymbol{d}_i^T \boldsymbol{y} z_i, i = 1, \ldots, p$. Inspired by [29, 32, 39], for any $Z \subseteq Z^0$ and $\boldsymbol{d}_i^T \boldsymbol{y} \in [L_i, U_i], i = 1, \ldots, p$, by using the concave envelope of bilinear function, it can be derived that the following relationships hold

$$h_i \leq l_i \boldsymbol{d}_i^T \boldsymbol{y} + U_i z_i - l_i U_i, h_i \leq u_i \boldsymbol{d}_i^T \boldsymbol{y} + L_i z_i - u_i L_i, i = 1, \ldots, p.$$

Then, the linear relaxation problem of EP(Z) is formulated.

$$(\text{LRP(Z)}) : \begin{cases} \max \varphi(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{h}) = \sum_{i=1}^{p} [h_i + (d_{0i}\boldsymbol{c}_i + c_{0i}\boldsymbol{d}_i)^T \boldsymbol{y} + c_{0i}d_{0i}] \\ \text{s.t.} \quad h_i \leq l_i \boldsymbol{d}_i^T \boldsymbol{y} + U_i z_i - l_i U_i, i = 1, \ldots, p, \\ \qquad h_i \leq u_i \boldsymbol{d}_i^T \boldsymbol{y} + L_i z_i - u_i L_i, i = 1, \ldots, p, \\ z_i = \boldsymbol{c}_i^T \boldsymbol{y}, i = 1, \ldots, p, \\ \boldsymbol{z} \in Z, \\ \boldsymbol{d}_i^T \boldsymbol{y} \in [L_i, U_i], i = 1, \ldots, p, \\ \boldsymbol{y} \in \mathbb{Y}. \end{cases}$$

Apparently, the optimal value of LRP(Z) can provide an upper bound for that of EP(Z).

Remark 1 $\min\{l_i \boldsymbol{d}_i^T \boldsymbol{y} + U_i - l_i U_i, u_i \boldsymbol{d}_i^T \boldsymbol{y} z_i + L_i z_i - u_i L_i\} - \boldsymbol{d}_i^T \boldsymbol{y} z_i \to 0, i = 1, \ldots, p$, when $(u_i - l_i) \to 0$ for $i = 1, \ldots, p$. The proof is similar to Yin et al. [31], in which the detail is presented in Theorem 1.

3 The Self-Adjustment Branching Rule

In this subsection, the proposal implements a self-adjustment branching rule for the algorithm in order to expedite the attainment of an optimal solution of LMP. In [29, 30, 32–35, 38], the standard bisection branching rule is used and it can be described as follows:

(i) Let $q = \arg\max\{u_i - l_i | i = 1, 2, \ldots, p\}$.
(ii) Z is subdivided into two p-dimensional sub-rectangles Z_1 and Z_2, i.e.,

$$Z_1 = \left\{ \boldsymbol{z} \in \mathbb{R}^p | l_i \le z_i \le u_i, i = 1, 2, \ldots, p, i \ne q, l_q \le z_q \le \frac{(l_q + u_q)}{2} \right\},$$
$$Z_2 = \left\{ \boldsymbol{z} \in \mathbb{R}^p | l_i \le z_i \le u_i, i = 1, 2, \ldots, p, i \ne q, \frac{(l_q + u_q)}{2} \le z_q \le u_q \right\}.$$

Using the standard bisection branching rule, the optimal solution of LRP(Z) may still be the optimal solution to LRP(Z_1) or LRP(Z_2), so that the upper bound of LMP(Z) is not updated during the execution of the algorithm. The current focus is on devising branching strategies that ensure a continuous update of the upper bound for the optimal value to LMP.

Lemma 1 *For any $Z \subseteq Z^0$, assume that $(\boldsymbol{y}^*, \boldsymbol{z}^*, \boldsymbol{h}^*)$ is an optimal solution of LRP(Z). For any i, if $z_i^* = l_i$ or $z_i^* = u_i$ or $u_i - l_i = 0$ or $\boldsymbol{d}_i^T \boldsymbol{y}^* = L_i$ or $\boldsymbol{d}_i^T \boldsymbol{y}^* = U_i$ or $U_i - L_i = 0$, we have $h_i^* = \boldsymbol{d}_i^T \boldsymbol{y}^* z_i^*$.*

Proof If $z_i^* = l_i$, then $h_i^* \le l_i \boldsymbol{d}_i^T \boldsymbol{y}^* + U_i z_i^* - l_i U_i = \boldsymbol{d}_i^T \boldsymbol{y}^* z_i^*$.

If $z_i^* = u_i$, then $h_i^* \le u_i \boldsymbol{d}_i^T \boldsymbol{y}^* + L_i z_i^* - u_i L_i = \boldsymbol{d}_i^T \boldsymbol{y}^* z_i^*$.

If $u_i - l_i = 0$, assume that $u_i = l_i = z_i^*$, then $h_i^* \le l_i \boldsymbol{d}_i^T \boldsymbol{y}^* + U_i z_i^* - l_i U_i = \boldsymbol{d}_i^T \boldsymbol{y}^* z_i^*$. and $h_i^* \le u_i \boldsymbol{d}_i^T \boldsymbol{y}^* + L_i z_i^* - u_i L_i = \boldsymbol{d}_i^T \boldsymbol{y}^* z_i^*$.

In the same way, if $\boldsymbol{d}_i^T \boldsymbol{y}^* = L_i$ or $\boldsymbol{d}_i^T \boldsymbol{y}^* = U_i$ or $U_i - L_i = 0$, then $h_i^* = \boldsymbol{d}_i^T \boldsymbol{y}^* z_i^*$. □

For any $Z \subseteq Z^0$, assume that $(\boldsymbol{y}^*, \boldsymbol{z}^*, \boldsymbol{h}^*)$ is an optimal solution of LRP(Z). We choose $\xi = \arg\max\{\min\{l_i \boldsymbol{d}_i^T \boldsymbol{y}^* + U_i z_i^* - l_i U_i,\ u_i \boldsymbol{d}_i^T \boldsymbol{y}^* + L_i z_i^* - u_i L_i\} - \boldsymbol{d}_i^T \boldsymbol{y}^* z_i^* \,|\, i = 1, 2, \ldots, p\}$ as the branching direction. According to the above Lemma 1, for any $i = 1, \ldots, p$, we can get $\min\{l_i \boldsymbol{d}_i^T \boldsymbol{y}^* + U_i z_i^* - l_i U_i,\ u_i \boldsymbol{d}_i^T \boldsymbol{y}^* + L_i z_i^* - u_i L_i\} - \boldsymbol{d}_i^T \boldsymbol{y}^* z_i^* = 0$ when $z_i^* = l_i$, $z_i^* = u_i$, $u_i - l_i = 0$, $\boldsymbol{d}_i^T \boldsymbol{y}^* = L_i$, $\boldsymbol{d}_i^T \boldsymbol{y}^* = U_i$, and $U_i - L_i = 0$, respectively, which will not be chosen for further division. Thus, Z is divided into Z^1 and Z^2 along $[l_\xi, u_\xi]$ at point r, as follows:

$$Z^1 = \{\boldsymbol{z} \in \mathbb{R}^p | l_i \le z_i \le u_i, i = 1, 2, \ldots, p, i \ne \xi, l_\xi \le z_\xi \le r\}, \tag{3.1}$$
$$Z^2 = \{\boldsymbol{z} \in \mathbb{R}^p | l_i \le z_i \le u_i, i = 1, 2, \ldots, p, i \ne \xi, r \le z_\xi \le u_\xi\}, \tag{3.2}$$

in which $r \in (l_\xi, u_\xi)$ is denoted as the branching point. Note that the inequality constraint of two sub-rectangles corresponding to the ξ direction is expressed as

$$h_\xi^* \le l_\xi \boldsymbol{d}_\xi^T \boldsymbol{y}^* + U_\xi z_\xi^* - l_\xi U_\xi, \tag{3.3}$$

$$h_\xi^* \le r \boldsymbol{d}_\xi^T \boldsymbol{y}^* + L_\xi z_\xi^* - r L_\xi, \tag{3.4}$$

$$h_\xi^* \le r \boldsymbol{d}_\xi^T \boldsymbol{y}^* + U_\xi z_\xi^* - r U_\xi, \tag{3.5}$$

$$h_\xi^* \le u_\xi \boldsymbol{d}_\xi^T \boldsymbol{y}^* + L_\xi z_\xi^* - u_\xi L_\xi. \tag{3.6}$$

Based on the above discussions, if the optimal solution $(\boldsymbol{y}^*, \boldsymbol{z}^*, \boldsymbol{h}^*)$ of LRP(Z) is cut off by dividing along $[l_\xi, u_\xi]$ at point r, i.e., $h_\xi^* > r \boldsymbol{d}_\xi^T \boldsymbol{y}^* + L_\xi z_\xi^* - r L_\xi$ and $h_\xi^* > r \boldsymbol{d}_\xi^T \boldsymbol{y}^* + U_\xi z_\xi^* - r U_\xi$ hold, so that we calculate and obtain

$$r < \frac{h_\xi^* - L_\xi z_\xi^*}{\boldsymbol{d}_\xi^T \boldsymbol{y}^* - L_\xi} \triangleq r1, \tag{3.7}$$

$$r > \frac{h_\xi^* - U_\xi z_\xi^*}{\boldsymbol{d}_\xi^T \boldsymbol{y}^* - U_\xi} \triangleq r2. \tag{3.8}$$

Combining $l_\xi, u_\xi, r1$ and $r2$, the selection of r is determined by the following Theorem 2.

Theorem 2 *For any $Z \subseteq Z^0$, assume that $(\boldsymbol{y}^*, \boldsymbol{z}^*, \boldsymbol{h}^*)$ is the optimal solution of LRP(Z). If $z_\xi^* \in (l_\xi, u_\xi)$, $(\boldsymbol{y}^*, \boldsymbol{z}^*, \boldsymbol{h}^*)$ is not the optimal solution of LRP(Z^1) and LRP(Z^2), where Z^1 and Z^2 are given in (3.1) and (3.2), respectively, and r in Z^1 and Z^2 is given as follows:*

$$r = \begin{cases} w, & if\ w \in (r1, r2), \\ r2 + \bar{\varepsilon}, & if\ w \le r2, \\ r1 - \bar{\varepsilon}, & if\ w \ge r1, \end{cases} \tag{3.9}$$

with (3.7), (3.8), $0 < \bar{\varepsilon} < r1 - r2$ and $w = \frac{1}{2}(u_\xi - l_\xi)$. Otherwise, let $r = z_\xi^*$, we get $h_i^* = \boldsymbol{d}_i^T \boldsymbol{y}^* z_i^*$, then the ξ-th edge of Z^s $(s = 1, 2)$ will not be chosen for further division.

Proof From (3.7) and (3.8), we have

$$r1 - r2 = \frac{h_\xi^* - L_\xi z_\xi^*}{\boldsymbol{d}_\xi^T \boldsymbol{y}^* - L_\xi} - \frac{h_\xi^* - U_\xi z_\xi^*}{\boldsymbol{d}_\xi^T \boldsymbol{y}^* - U_\xi} \ge 0,$$

$$r1 - u_\xi = \frac{h_\xi^* - L_\xi z_\xi^*}{\boldsymbol{d}_\xi^T \boldsymbol{y}^* - L_\xi} - u_\xi = \frac{h_\xi^* - L_\xi z_\xi^* - u_\xi \boldsymbol{d}_\xi^T \boldsymbol{y}^* + u_\xi L_\xi}{\boldsymbol{d}_\xi^T \boldsymbol{y}^* - L_\xi} \le 0,$$

$$r2 - l_\xi = \frac{h_\xi^* - U_\xi z_\xi^*}{\boldsymbol{d}_\xi^T \boldsymbol{y}^* - U_\xi} - l_\xi = \frac{h_\xi^* - U_\xi z_\xi^* - l_\xi \boldsymbol{d}_\xi^T \boldsymbol{y}^* + l_\xi U_\xi}{\boldsymbol{d}_\xi^T \boldsymbol{y}^* - U_\xi} \ge 0,$$

where the above inequalities follow from $L_\xi < \boldsymbol{d}_\xi^T \boldsymbol{y} < U_\xi$, (3.6) and (3.3), respectively. Thus, we obtain

$$l_\xi \le r2 < r1 \le u_\xi.$$

For $l_\xi < z_\xi^* < u_\xi$, we have the following two cases:

(i). If $r2 \le w \le r1$, then $r = w$.

Since $r > r2$, it follows that

$$h_\xi^* - r\boldsymbol{d}_\xi^T \boldsymbol{y}^* - U_\xi z_\xi^* + rU_\xi > h_\xi^* - U_\xi z_\xi^* - r2(\boldsymbol{d}_\xi^T \boldsymbol{y}^* - U_\xi) = 0. \tag{3.10}$$

This contradicts (3.5), i.e., $(\boldsymbol{y}^*, \boldsymbol{z}^*, \boldsymbol{h}^*)$ is not the optimal solution of LRP(Z^2).

Similarly, since $r < r1$, it follows that

$$h_\xi^* - r\boldsymbol{d}_\xi^T \boldsymbol{y}^* - L_\xi z_\xi^* + rL_\xi > h_\xi^* - L_\xi z_\xi^* - r1(\boldsymbol{d}_\xi^T \boldsymbol{y}^* - L_\xi) = 0. \tag{3.11}$$

This contradicts (3.4), i.e., $(\boldsymbol{y}^*, \boldsymbol{z}^*, \boldsymbol{h}^*)$ is not the optimal solution of LRP(Z^1).

(ii). If $w \notin (r2, r1)$, then $w \le r2$ or $w \ge r1$.

If $w \le r2$, then $r = r2 + \bar{\varepsilon}$. Since $r2 \le r = r2 + \bar{\varepsilon} \le r1$, $(\boldsymbol{y}^*, \boldsymbol{z}^*, \boldsymbol{h}^*)$ is not the optimal solution of LRP(Z^2) by (3.10).

Similarly, if $w \ge r1$, then $r = r1 - \bar{\varepsilon}$. Thus, $(\boldsymbol{y}^*, \boldsymbol{z}^*, \boldsymbol{h}^*)$ is not the optimal solution of LRP(Z^1) by (3.11).

If $z_\xi^* = l_\xi$ or $z_\xi^* = u_\xi$, then $r = z_\xi^*$, such that $(\boldsymbol{y}^*, \boldsymbol{z}^*, \boldsymbol{h}^*)$ is not the optimal solution of LRP(Z^1) and LRP(Z^2). In addition,

$$\begin{aligned}
&l_\xi \boldsymbol{d}_\xi^T \boldsymbol{y}^* + U_\xi z_\xi^* - l_\xi U_\xi - h_\xi^* = 0,\ r\boldsymbol{d}_\xi^T \boldsymbol{y}^* + L_\xi z_\xi^* - rL_\xi - h_\xi^* = 0,\\
&r\boldsymbol{d}_\xi^T \boldsymbol{y}^* + U_\xi z_\xi^* - rU_\xi - h_\xi^* = 0,\ u_\xi \boldsymbol{d}_\xi^T \boldsymbol{y}^* + L_\xi z_\xi^* - u_\xi L_\xi - h_\xi^* = 0,
\end{aligned}$$

i.e., the ξ-th edge of Z^s $(s = 1, 2)$ will not be chosen for further division. □

By Theorem 2, the closer r is to w, the smaller the approximation error between LRP and LMP over Z^1 and Z^2. The reasons are given below: let

$$\begin{aligned}
&h(z_\xi) = \boldsymbol{d_i}^T \boldsymbol{y} z_\xi,\\
&h^{11}(z_\xi) = l_\xi \boldsymbol{d_i}^T \boldsymbol{y} + U_\xi z_\xi - l_\xi U_\xi, h^{12}(z_\xi) = r\boldsymbol{d_i}^T \boldsymbol{y} + L_\xi z_\xi^* - rL_\xi,\\
&h^{21}(z_\xi) = r\boldsymbol{d_i}^T \boldsymbol{y} + U_\xi z_\xi - rU_\xi, h^{22}(z_\xi) = u_\xi \boldsymbol{d_i}^T \boldsymbol{y} + L_\xi z_\xi - u_\xi L_\xi.
\end{aligned}$$

The region between $h(z_\xi)$ and $h^{11}(z_\xi), h^{12}(z_\xi), h^{21}(z_\xi), h^{22}(z_\xi)$ is determined by

$$\begin{aligned}
S(r) = &\int_{l_\xi}^{r} [h^{11}(z_\xi) - h(z_\xi)]dz_\xi + \int_{r}^{u_\xi} [h^{12}(z_\xi) - h(z_\xi)]dz_\xi + \int_{l_\xi}^{r} [h^{21}(z_\xi) - h(z_\xi)]dz_\xi\\
&+ \int_{r}^{u_\xi} [h^{22}(z_\xi) - h(z_\xi)]dz_\xi\\
= &(U_\xi - L_\xi)r^2 + (u_\xi - l_\xi)(L_\xi - U_\xi)r + u_\xi^2(z_\xi - L_\xi) - l_\xi^2(z_\xi - U_\xi).
\end{aligned}$$

It can obtain the minimum of $S(r)$ at $r = \frac{1}{2}(u_\xi - l_\xi)$, so that the chosen of r is given in (3.9). On the basis of the above discussions, the self-adjusting branching rule is summarized as follows:

The self-adjustable branching rule
***Step* 1 :** Let $\xi = \arg\max\{\min\{l_i \boldsymbol{d}_i^T \boldsymbol{y}^* + U_i z_i^* - l_i U_i,\ u_i \boldsymbol{d}_i^T \boldsymbol{y}^* + L_i z_i^* - u_i L_i\} - \boldsymbol{d}_i^T \boldsymbol{y}^* z_i^* \mid i = 1, 2, \ldots, p\}$.
***Step* 2 :** If $l_\xi < z_\xi^* < u_\xi$, then choose r by (3.9). otherwise, let $r = z_\xi^*$.
***Step* 3 :** Z is subdivided into two p-dimensional sub-rectangles Z^1 and Z^2, which are given by (3.1) and (3.2), respectively.

Based on Theorem 2, the optimal solution of LRP(Z) can be cut off by the self-adjusting branching rule. The implication is that each iteration of the algorithm may enhance the upper bound for the optimal value to LMP. In contrast, using the standard bisection branching rule, the optimal solution of LRP corresponding to the divided rectangle may be still the optimal solution of the sub-rectangle, thereby contributing to an increase in computational cost.

4 Algorithm, Convergence and Complexity

In this section, based on LRP and branching rule, we present a self-adjustable branch-and-bound (SABB) algorithm for solving LMP. By subsequently subdividing the initial image space rectangle and solving a series of linear relaxation problems, we establish the global convergence of the algorithm and estimate its complexity.

4.1 SABB Algorithm

The global optimal solution of LMP is achieved through the introduction of SABB algorithm based on the linear relaxation program, self-adjustable branching rule and branch-and-bound framework.

The proposed branching process is conducted on the image space using the self-adjustable branching rule, which is different from other branching methods based on the original decision variables such as the algorithms in [29, 30, 34]. Specifically, the branching process of this text takes place in image space R^p of the affine function $\boldsymbol{c_i}^T \boldsymbol{x}, i = 1, \ldots, p$. These distinctions imply that the proposed algorithm may be even better in economize on the required computations if $p \ll n$.

At stage k of SABB algorithm, assume that

$$Z^k = \left\{ z \in \mathbb{R}^p | l_i^k \le z_i \le u_i^k, i = 1, \ldots, p \right\} \subseteq Z^0. \tag{4.1}$$

Thus, Z^k is divided into two rectangles Z^{k1} and Z^{k2} by the self- adjustable branching rule such that $Z^k = Z^{k1} \cup Z^{k2}$.

Based on the above discussions, the basic steps of SABB algorithm for globally solving LMP are summarized as follows.

SABB algorithm statement:

Step 0: Given the error tolerance $\varepsilon > 0$, and an initial rectangle Z^0. Solve LRP(Z^0) to obtain its optimal solution $(\boldsymbol{y}^0, \boldsymbol{z}^0, \boldsymbol{h}^0)$ and optimal value $\varphi(\boldsymbol{y}^0, \boldsymbol{z}^0, \boldsymbol{h}^0)$. Set $UB_0 = \varphi(\boldsymbol{y}^0, \boldsymbol{z}^0, \boldsymbol{h}^0), \bar{h_i}^0 = \boldsymbol{d}_i^T \boldsymbol{y}^0 z_i^0, i = 1, 2, \ldots, p, LB_0 = \phi(\boldsymbol{y}^0, \boldsymbol{z}^0, \bar{\boldsymbol{h}}^0)$. If $UB_0 - LB_0 \leq \varepsilon$, then the algorithm stops, and $\boldsymbol{y}^0$ is a global ε-optimal solution for LMP. Otherwise, let $X = \{(\boldsymbol{y}^0, \boldsymbol{z}^0, \bar{\boldsymbol{h}}^0)\}$, $k = 0$, set $T_0 = \{Z^0\}$.

Step 1: Using the self-adjustable branching rule to subdivide Z^k into two new sub-rectangles Z^{k1}, Z^{k2} and set $T = \{Z^{k1}, Z^{k2}\}$.

Step 2: For each $Z^{ks}(s = 1, 2)$, solve LRP(Z^{ks}) to obtain the optimal solution $(\boldsymbol{y}^{ks}, \boldsymbol{z}^{ks}, \boldsymbol{h}^{ks})$ and optimal value $\varphi(\boldsymbol{y}^{ks}, \boldsymbol{z}^{ks}, \boldsymbol{h}^{ks})$. Set $UB(Z^{ks}) = \varphi(\boldsymbol{y}^{ks}, \boldsymbol{z}^{ks}, \boldsymbol{h}^{ks})$, let $\bar{h}_i^{ks} = \boldsymbol{d}_i^T \boldsymbol{y}^{ks} z_i^{ks}$, $i = 1, 2, \ldots, p$, $X = X \bigcup \{(\boldsymbol{y}^{ks}, \boldsymbol{z}^{ks}, \bar{\boldsymbol{h}}^{ks})\}$. If $LB_k > UB(Z^{ks})$, set $T_k = T_k \setminus Z^{ks}$. Let $T_k = (T_k \backslash Z^k) \bigcup T$. Update lower bound $LB_k = \max_{(\boldsymbol{y},\boldsymbol{z},\boldsymbol{h}) \in X} \phi(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{h})$, and set $(\boldsymbol{y}^k, \boldsymbol{z}^k, \boldsymbol{h}^k) = \arg\max_{(\boldsymbol{y},\boldsymbol{z},\boldsymbol{h}) \in X} \phi(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{h})$.

Step 3: Set $T_{k+1} = \{Z \mid UB(Z) - LB_k > \varepsilon, Z \in T_k\}$. If $T_{k+1} = \emptyset$, then terminate: $\boldsymbol{y}^k$ is a global ε-optimal solution for LMP. Otherwise, select the rectangle Z^{k+1} such that $Z^{k+1} = \arg\max_{Z \in T_{k+1}} UB(Z)$, set $k = k + 1$, and return to Step 1.

4.2 Convergence of SABB Algorithm

In this subsection, we discuss the global convergence of SABB algorithm.

Theorem 3 *Given $\varepsilon \geq 0$, if SABB algorithm terminates finitely, then it returns a global ε-optimal solution of LMP; otherwise, an infinite sequence $\{\boldsymbol{y}^k\}$ is generated, and every accumulation point of that is a global optimal solution for LMP.*

Proof If the presented algorithm terminates finitely, without loss of generality, suppose it terminates at kth iteration. Then we have

$$UB_k - LB_k \leq \varepsilon. \tag{4.2}$$

According to Step 2 of the algorithm,

$$LB_k = \phi(\boldsymbol{y}^k, \boldsymbol{z}^k, \bar{\boldsymbol{h}}^k) = f(\boldsymbol{y}^k), \tag{4.3}$$

By (4.2) and (4.3), we can get

$$UB_k - f(\boldsymbol{y}^k) \leq \varepsilon. \tag{4.4}$$

Let f^* be the global optimal value of LMP. Combining (4.2)–(4.4), we have

$$f^* - \varepsilon \leq UB_k - \varepsilon \leq LB_k = f(\boldsymbol{y}^k).$$

So, $\boldsymbol{y}^k$ is a global ε-optimal solution of LMP.

If the algorithm is infinite, it can generate an infinitely nested sequence of rectangles $\{Z^k\}$, such that $u_i^k - l_i^k \to 0$ as $k \to \infty$ for $i = 1, 2, \ldots, p$. It also generates the optimal solution sequence $\{(\boldsymbol{y}^k, \boldsymbol{z}^k, \boldsymbol{h}^k)\}$ to LRP(Z^k). Let $\bar{h}_i^k = \boldsymbol{d_i} \boldsymbol{y}^k z_i^k, i = 1, \ldots, p,$

then $(\boldsymbol{y}^k, \boldsymbol{z}^k, \bar{\boldsymbol{h}}^k)$ is a feasible solution sequence for EP(Z^k). Since the feasible region of EP(Z^k) is bounded, without loss of generality, assume that $\lim_{k\to\infty} \boldsymbol{y}^k = \boldsymbol{y}^*$, then we have that

$$\lim_{k\to\infty} z_i^k = \lim_{k\to\infty} \boldsymbol{c}_i^T \boldsymbol{y}^k = \boldsymbol{c}_i^T \boldsymbol{y}^* \triangleq z_i^*, i = 1, 2, \ldots, p$$
$$\lim_{k\to\infty} \bar{h}_i^k = \lim_{k\to\infty} \boldsymbol{d}_i^T \boldsymbol{y}^k z_i^k = \boldsymbol{d}_i^T \boldsymbol{y}^* z_i^* \triangleq h_i^*, i = 1, 2, \ldots, p.$$

According to the definition of LB_k and the continuity of the function ϕ, we have

$$\lim_{k\to\infty} LB_k = \lim_{k\to\infty} f(\boldsymbol{y}^k) = \lim_{k\to\infty} \phi(\boldsymbol{y}^k, \boldsymbol{z}^k, \bar{\boldsymbol{h}}^k) = \phi(\boldsymbol{y}^*, \boldsymbol{z}^*, \boldsymbol{h}^*) = f(\boldsymbol{y}^*). \quad (4.5)$$

From the above results and $l_i^k \le z_i^k = \boldsymbol{c}_i^T \boldsymbol{y}^k \le u_i^k$, it follows that

$$\lim_{k\to\infty} l_i^k = \lim_{k\to\infty} z_i^k = \lim_{k\to\infty} u_i^k = z_i^*.$$

Additionally, from the inequality constraints $h_i^k \le l_i^k \boldsymbol{d}_i^T \boldsymbol{y}^k + U_i z_i^k - l_i^k U_i$ and $h_i^k \le u_i^k \boldsymbol{d}_i^T \boldsymbol{y}^k + L_i z_i^k - u_i^k L_i$, for any i, we have

$$\lim_{k\to\infty} h_i^k \le \lim_{k\to\infty} (l_i^k \boldsymbol{d}_i^T \boldsymbol{y}^k + U_i z_i^k - l_i^k U_i) = z_i^* \boldsymbol{d}_i^T \boldsymbol{y}^* + U_i z_i^* - z_i^* U_i = z_i^* \boldsymbol{d}_i^T \boldsymbol{y}^*,$$
$$\lim_{k\to\infty} h_i^k \le \lim_{k\to\infty} (u_i^k \boldsymbol{d}_i^T \boldsymbol{y}^k + L_i z_i^k - u_i^k L_i) = z_i^* \boldsymbol{d}_i^T \boldsymbol{y}^* + L_i z_i^* - z_i^* L_i = z_i^* \boldsymbol{d}_i^T \boldsymbol{y}^*.$$

This implies that

$$\begin{aligned}
\lim_{k\to\infty} UB_k &= \lim_{k\to\infty} \varphi(\boldsymbol{y}^k, \boldsymbol{z}^k, \boldsymbol{h}^k) \\
&= \lim_{k\to\infty} \sum_{i=1}^{p} (h_i^k + (d_{0i}\boldsymbol{c_i} + c_{0i}\boldsymbol{d_i})^T \boldsymbol{y}^k + c_{0i}d_{0i}) \\
&\le \sum_{i=1}^{p} (\boldsymbol{d}_i^T \boldsymbol{y}^* z_i^* + (d_{0i}\boldsymbol{c}_i + c_{0i}\boldsymbol{d}_i)^T \boldsymbol{y}^* + c_{0i}d_{0i}) \\
&= \sum_{i=1}^{p} (h_i^* + (d_{0i}\boldsymbol{c}_i + c_{0i}\boldsymbol{d}_i)^T \boldsymbol{y}^* + c_{0i}d_{0i}) \\
&= \phi(\boldsymbol{y}^*, \boldsymbol{z}^*, \boldsymbol{h}^*).
\end{aligned}$$

Further, combining (4.5) with $LB_k \le UB_k$, we have

$$\lim_{k\to\infty} LB_k = \lim_{k\to\infty} UB_k. \quad (4.6)$$

Based upon the structure of SABB algorithm, then it must have

$$LB_k \le f^* \le UB_k, \quad k = 0, 1, 2, \ldots. \quad (4.7)$$

From (4.5)-(4.7), we can get

$$\lim_{k\to\infty} LB_k = \lim_{k\to\infty} UB_k = f^* = f(\boldsymbol{y}^*). \tag{4.8}$$

Since $(\boldsymbol{y}^*, \boldsymbol{z}^*, \boldsymbol{h}^*)$ is a feasible solution of EP(Z^k), according to (4.8), $(\boldsymbol{y}^*, \boldsymbol{z}^*, \boldsymbol{h}^*)$ is an optimal solution of EP(Z^k). Furthermore, by using Theorem 1 and Theorem 2, it follows that $\boldsymbol{y}^*$ is a global optimal solution of LMP, i.e., every accumulation point of the sequence $\{\boldsymbol{y^k}\}$ is a global optimal solution for LMP. The proof is completed □

4.3 Computational Complexity of SABB Algorithm

In order to estimate the maximum iterations of SABB algorithm, we analyze its computational complexity. To this end, we define the size $\delta(Z)$ for the rectangle (2.1) as

$$\delta(Z) = \max\{u_j - l_j \mid j = 1, \ldots, p\}, \tag{4.9}$$

and for convenience, we define

$$\mu = \max\{U_j - L_j \mid j = 1, \ldots, p\}. \tag{4.10}$$

Lemma 2 *Given $\varepsilon \geq 0$, for any $Z \in Z^0$ and any feasible solution $(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{h})$ of LRP(Z), if $\delta(Z) \leq \varepsilon/p\mu$, then we have*

$$|\varphi(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{h}) - \phi(\boldsymbol{y}, \boldsymbol{z}, \bar{\boldsymbol{h}})| \leq \varepsilon,$$

in which $\bar{h}_i = \boldsymbol{d}_i^T \boldsymbol{y} z_i, i = 1, \ldots, p$.

Proof For any feasible solution $(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{h})$ for LRP(Z), let $\bar{h}_i = \boldsymbol{d}_i^T \boldsymbol{y} z_i, i = 1, \ldots, p$, it is obvious that $(\boldsymbol{y}, \boldsymbol{z}, \bar{\boldsymbol{h}})$ is a feasible solution for EP over Z. From Theorem 1 and Theorem 2, we have $\phi(\boldsymbol{y}, \boldsymbol{z}, \bar{\boldsymbol{h}}) = f(\boldsymbol{y})$. If $\delta(Z) \leq \varepsilon/p\mu$ for any sufficiently small positive number ε, we can obtain

$$\begin{aligned}
|\varphi(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{h}) - f(\boldsymbol{y})| &= |\varphi(\boldsymbol{y}, \boldsymbol{z}, \boldsymbol{h}) - \phi(\boldsymbol{y}, \boldsymbol{z}, \bar{\boldsymbol{h}})| \\
&= \sum_{i=1}^{p} (h_i - \bar{h}_i) \\
&\leq \sum_{i=1}^{p} \left[\min\{l_i \boldsymbol{d}_i^T \boldsymbol{y} + U_i z_i - l_i U_i, u_i \boldsymbol{d}_i^T \boldsymbol{y} + L_i z_i - u_i L_i\} - \boldsymbol{d}_i^T \boldsymbol{y} z_i\right] \\
&= \sum_{i=1}^{p} \min\{(z_i - l_i)(U_i - \boldsymbol{d}_i^T \boldsymbol{y}), (u_i - z_i)(\boldsymbol{d}_i^T \boldsymbol{y} - L_i)\} \\
&\leq p\mu\delta(Z) \\
&\leq \varepsilon.
\end{aligned}$$

□

Theorem 4 *Given the convergence tolerance $\varepsilon \in (0, 1)$, SABB algorithm can find a global ε-optimal solution to LMP in at most*

$$N = p.\left\lceil \log_2 \frac{p\mu\delta(Z^0)}{\varepsilon} \right\rceil$$

iterations, where $\delta(Z^0)$ and μ are given by (4.9) and (4.10), respectively.

Proof Without loss of generality, assume that the convergence tolerance $\varepsilon \in (0, 1)$ at the initializing step and that the sub-rectangle Z is selected for partitioning in step 1 of SABB algorithm at every iteration. After $k.p$ iterations, we have

$$\delta(Z) \leq \frac{1}{2^k}\delta(Z^0).$$

From Lemma 1, if

$$\frac{1}{2^k}\delta(Z^0) \leq \frac{\varepsilon}{p\mu},$$

i.e.,

$$k \geq \log_2 \frac{p\mu\delta(Z^0)}{\varepsilon},$$

we can obtain $|\varphi(\boldsymbol{x}, \boldsymbol{y}, z, \boldsymbol{h}) - \phi(\boldsymbol{x}, \boldsymbol{y}, z, \bar{\boldsymbol{h}})| \leq \varepsilon$. Therefore, after at most

$$p.\lceil \log_2 \frac{p\mu\delta(Z^0)}{\varepsilon} \rceil$$

iterations, we can follow that

$$\begin{aligned} 0 &\leq \phi(\boldsymbol{y}^*, z^*, \boldsymbol{h}^*) - \phi(\boldsymbol{y}, z, \bar{\boldsymbol{h}}) \\ &\leq |\varphi(\boldsymbol{y}, z, \boldsymbol{h}) - \phi(\boldsymbol{y}, z, \bar{\boldsymbol{h}})| \\ &\leq \varepsilon, \end{aligned}$$

where $(\boldsymbol{y}^*, z^*, \boldsymbol{h}^*)$ is the optimal solution of EP. Therefore, $\boldsymbol{y}^*$ is the optimal solution of LMP. In step 2 of SABB algorithm, $(\boldsymbol{y}^k, z^k, \bar{\boldsymbol{h}}^k)$ is the best currently known feasible solution, and we also note that $\{\phi(\boldsymbol{y}^k, z^k, \bar{\boldsymbol{h}}^k)\}$ is a increasing sequence satisfying

$$\phi(\boldsymbol{y}^k, z^k, \bar{\boldsymbol{h}}^k) \geq \phi(\boldsymbol{y}, z, \bar{\boldsymbol{h}}).$$

Therefore, we have

$$\phi(\boldsymbol{y}^*, z^*, \boldsymbol{h}^*) - \phi(\boldsymbol{y}^k, z^k, \bar{\boldsymbol{h}}^k) \leq \phi(\boldsymbol{y}^*, z^*, \boldsymbol{h}^*) - \phi(\boldsymbol{y}, z, \bar{\boldsymbol{h}}) \leq \varepsilon,$$

which implies that $f(\boldsymbol{y}^*) - f(\boldsymbol{y}^k) \leq \varepsilon$. When SABB algorithm terminates, $\boldsymbol{y}^k$ is a global ε-optimal solution to LMP, and the proof is completed. □

Table 1 Notations in numerical instances

Item	Description
LRP	The linear relaxation method in this paper
LRP1(LRP2)	Two linear relaxation methods in [21]
$RP_{[*]}$	The relaxation method in reference [*]
SABB	BB algorithm with self-adjustable branching rule and choosing the branching direction where the relaxation is tighter direction of approximation error
BRBB	BB algorithm with bisection rule
CRBB	BB algorithm with combination rule
SABB-L	BB algorithm with self-adjustable branching rule and choosing the longest edge of rectangles as the branching direction
Opt.val	The average optimal value obtained by the relevant algorithm
Iter	The average number of the algorithm iterations
CPU	The average CPU time calculated by the algorithm in seconds
Gap	The difference in corresponding numerical comparison of two algorithms
Gap(%)	The percentage of Gap.CPU to BBB.CPU or Gap.Iter to BBB.Iter
"…/-"	The method fails to find the optimal solution in 3600 s/10 s at all cases
Bold font	The best result in a set of algorithm comparisons

5 Numerical Experiments

In this section, the effectiveness of SABB algorithm is validated by numerical computation. All of algorithms are coded in MATLAB (2018b), then it runs on in a computer with Intel(R) Core(TM)i9-13900HX CPU(2.20GHz). The linear and quadratic subproblem in the algorithms are solved by the *linprog* and *quadprog* in MATLAB, respectively.

Each (m, n, p) generates ten randomly instances, and their average results are obtained. The error tolerance ε and $\bar{\varepsilon}$ are set to 10^{-4} and the maximum CPU time is limited to 3600 s or 10 s in line with the computation requirement. The notations of computational results are listed in Table 1.

In computational experiments, we consider the following LMP:

$$(\mathrm{P}):\begin{cases}\max \sum_{i=1}^{p}\left(\sum_{j=1}^{n} c_{ij}x_j + c_{0i}\right)\left(\sum_{j=1}^{n} d_{ij}x_j + d_{0i}\right)\\ \text{s.t.} \quad \boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b},\end{cases}$$

where c_{ij}, d_{ij}, c_{0i} and d_{0i}, are randomly generated in $[-5, 5]$, and all elements of $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{R}^{m}$ are randomly generated in $[0.1, 1]$.

5.1 Numerical Comparison of Relaxation Methods

To evaluate the performance of the proposed linear relaxation method (LRP), the comparisons of LRP and the existing relaxation methods in [21, 29, 30, 32–35, 37] are performed.

Since the relaxation methods in [21], denoted as LRP1 and LRP2, are similar to LRP, their performances are first compared. As shown in Table 2, the upper bound values of LRP are smaller than those of LRP1 and LRP2, and the lower bound values of LRP are lager than those of LRP1 and LRP2. The CPU time spent by LRP is longer than those of LRP1 and LRP2 in most cases. These results mean that LRP provides a tighter bound than LRP1 and LRP2. Addtionally, the upper bound obtained by LRP is smaller than those obtained by the relaxation methods in [29, 30, 32–35, 37] for all randomly generated instances, as shown Table 3. These results suggest that LRP is superior to the relaxation methods in [29, 30, 32–35, 37].

5.2 Numerical Comparison of Branching Rules

In order to explore the influence of branching rules on the algorithm, the comparisons of the self-adjustable branching rule (SABB algorithm) and the combination rules (BRBB and CRBB algorithms) are performed.

Let $Z = \Pi_{i=1}^{p}[l_i, u_i]$ denote the divided rectangle, z^{optx} denote the corresponding optimal solution of the relaxation problem over Z, ξ denote the branching direction chosen, z_{ξ}^{*} denote the branching point chosen of the interval $[l_{\xi}, u_{\xi}]$ and z_{ξ}^{M} denote the medium point of the interval $[l_{\xi}, u_{\xi}]$, respectively. The bisection rule chooses $z_{\xi}^{*} = z_{\xi}^{M}$ as the the branching point in [29, 30, 32–35]. The combination rule in [21] chooses a linear combination of z^{optx} and z_{ξ}^{M} as its the branching point, i.e., $z_{\xi}^{*} = \alpha z^{optx} + (1-\alpha) z_{\xi}^{M}$, $\alpha \in [0, 1]$. Note that the bisection rule is the combination rule at $\alpha = 0$, denoted as BRBB algorithm.

Table 4 shows that both the CPU time and the number of iterations of SABB algorithm are lower than those in BRBB algorithm. When (m, p) is fixed, both the value of Gap.CPU and Gap.Iter grow with n grows. In contrast, both the value of Gap(%).CPU and Gap(%).Iter decrease with n increases. On the whole, the self-adjustable branching rule in branch-and-bound algorithm is better than the bisection branching rule (the combination rule at $\alpha = 0$) in branch-and-bound algorithm, which may be the result of the former can keep update the upper bound of LMP.

We further compare the performance of SABB and CRBB algorithms at $\alpha = 0.3, 0.5, 0.7$ and 1.0 (CRBB algorithms), as shown in Table 5. When $n \leq 50$, the CPU time spent by CRBB algorithm at $\alpha = 1.0$ is less than that of SABB algorithm and those of CRBB algorithms. However, SABB algorithm is better than CRBB algorithms when $n > 100$, which suggest that the self-adjustable branching rule has a better effect on the algorithm than the combination rules for large-scale test instances.

Table 2 Comparison results of LRP, LRP1 and LRP2 for solving P

(m, n, p)	Opt.val	Upper bound			Lower bound			CPU		
	SABB	LRP	LRP1	LRP2	LRP	LRP1	LRP2	LRP	LRP1	LRP2
(100, 10, 2)	1.8515	**2.7308**	3.7407	3.7407	**1.5814**	0.8035	0.8035	0.01	0.01	0.01
(100, 50, 2)	0.5481	**0.5587**	0.5691	0.5691	**0.5473**	0.5440	0.5440	0.05	0.05	0.05
(100, 100, 2)	0.5385	**0.5489**	0.5558	0.5558	**0.5372**	0.5342	0.5342	0.12	**0.11**	0.12
(100, 500, 2)	0.6199	**0.6390**	0.6533	0.6533	**0.6182**	0.6138	0.6138	1.40	1.40	1.40
(100, 1000, 2)	0.7380	**0.7611**	0.7779	0.7779	**0.7353**	0.7250	0.7250	**5.93**	**5.93**	5.96
(100, 2000, 2)	12.0430	**57.3285**	85.9304	85.9304	**10.6543**	− 1.8217	− 1.8217	35.39	35.37	**35.36**
(100, 5000, 2)	0.7697	**0.8010**	0.8226	0.8226	**0.7668**	0.7502	0.7502	**282.65**	284.28	284.77
(100, 10, 5)	1.3370	**1.3462**	1.3569	1.3569	1.3355	**1.3356**	**1.3356**	0.02	0.02	0.02
(100, 50, 5)	1.2389	**1.2670**	1.2898	1.2898	**1.2361**	1.2272	1.2272	0.06	0.06	**0.05**
(100, 100, 5)	1.6211	**1.6584**	1.6829	1.6829	**1.6169**	1.6158	1.6158	0.13	0.13	0.13
(100, 1000, 5)	1.6911	**1.7691**	1.8050	1.8050	**1.6842**	1.6758	1.6758	6.08	**6.05**	**6.05**
(100, 2000, 5)	1.7042	**1.7934**	1.8351	1.8351	**1.6977**	1.6816	1.6816	32.16	**32.13**	32.29
(100, 5000, 5)	1.7207	**1.8248**	1.8678	1.8678	**1.7094**	1.6918	1.6918	275.95	**275.83**	275.88
(100, 10, 8)	2.2495	**2.2599**	2.2883	2.2883	**2.2490**	2.2485	2.2485	**0.02**	**0.02**	0.03
(100, 50, 8)	2.3189	**2.3700**	2.4027	2.4027	2.3146	**2.3150**	**2.3150**	0.06	0.06	0.06
(100, 100, 8)	2.1485	**2.2242**	2.2851	2.2851	**2.1408**	2.1400	2.1400	0.14	0.14	0.14
(100, 500, 8)	2.3975	**2.5238**	2.5766	2.5766	**2.3888**	2.3824	2.3824	1.49	**1.48**	**1.48**
(100, 1000, 8)	2.3324	**2.4621**	2.5366	2.5366	**2.3199**	2.3111	2.3111	**6.13**	6.14	6.14

Table 2 continued

(m, n, p)	Opt.val	Upper bound			Lower bound			CPU		
	SABB	LRP	LRP1	LRP2	LRP	LRP1	LRP2	LRP	LRP1	LRP2
(100, 2000, 8)	2.5102	**2.6817**	2.7580	2.7580	**2.4942**	2.4706	2.4706	32.26	**32.13**	**32.13**
(100, 5000, 8)	2.7152	**2.8897**	2.9641	2.9641	**2.7002**	2.6775	2.6775	281.35	**280.61**	280.70
(100, 10, 10)	3.5929	**3.6204**	3.6904	3.6904	**3.5917**	3.5907	3.5907	**0.02**	**0.02**	0.03
(100, 50, 10)	3.4709	**3.5672**	3.6544	3.6544	**3.4645**	3.4613	3.4613	**0.06**	0.07	**0.06**
(100, 100, 10)	4.1984	**20.6580**	30.6404	30.6404	**3.1599**	−0.4298	−0.4298	0.16	0.16	0.16
(100, 500, 10)	3.9510	**4.2236**	4.3453	4.3453	**3.9363**	3.9279	3.9279	**1.48**	1.49	1.49
(100, 1000, 10)	3.7641	**4.0751**	4.2071	4.2071	**3.7320**	3.7211	3.7211	6.05	6.05	**6.03**
(100, 2000, 10)	3.8214	**4.1555**	4.2901	4.2901	**3.7909**	3.7624	3.7624	31.70	**31.65**	**31.65**

Table 3 Comparison results of LRP and the relaxation methods in [29, 30, 32–35, 37] for solving P

(m, n, p)	LRP	$\text{RP}_{[29]}$	$\text{RP}_{[32]}$	$\text{RP}_{[33]}$	$\text{RP}_{[35]}$	$\text{RP}_{[34]}$	$\text{RP}_{[30]}$	$\text{RP}_{[37]}$
(50, 10, 4)	**8.8151**	11.1451	11.1451	12.1033	12.1033	42.8032	11.0680	12.8215
(50, 20, 4)	**12.3516**	16.1044	16.1044	16.2217	16.2217	88.0050	18.9241	21.9847
(50, 50, 4)	**6.7778**	9.0112	9.0112	10.3111	10.3111	184.7954	18.1571	15.2453
(50, 100, 4)	**24.0113**	26.1039	26.1039	26.3286	26.3286	452.9088	49.8921	35.8749
(50, 200, 4)	**23.6022**	25.8058	25.8058	25.1658	25.1658	1012.4634	101.0590	43.2035
(50, 500, 4)	**25.4528**	31.3039	31.3039	30.3705	30.3705	2476.0094	270.4134	53.8162
(50, 1000, 4)	**15.6098**	21.3201	21.3201	22.2366	22.2366	4750.5518	554.8122	51.6692
(50, 2000, 4)	**28.3537**	32.9560	32.9560	32.5938	32.5938	10728.8025	1214.0283	66.2631
(50, 5000, 4)	**22.6648**	25.8755	25.8755	26.2461	26.2461	29118.7881	3206.6586	69.5985
(50, 10, 8)	**2.2981**	7.4345	7.4345	8.7899	8.7899	71.6775	6.5835	14.8596
(50, 20, 8)	**−0.3312**	5.3411	5.3411	5.8824	5.8824	149.2197	5.0808	17.5371
(50, 50, 8)	**8.2775**	11.7142	11.7142	11.5349	11.5349	301.5649	17.0451	27.6782
(50, 100, 8)	**28.5121**	33.2408	33.2408	31.3044	31.3044	842.8830	56.7530	61.4757
(50, 200, 8)	**33.4793**	37.5117	37.5117	35.9678	35.9678	1780.8751	105.0252	77.4916
(50, 500, 8)	**63.4785**	69.3231	69.3231	64.4620	64.4620	5340.6121	363.4346	137.8733
(50, 1000, 8)	**44.5000**	51.4594	51.4594	47.6899	47.6899	8928.7476	576.7886	118.7386
(50, 2000, 8)	**41.1020**	46.7290	46.7290	43.5441	43.5441	19173.9982	1246.4685	129.5789

Table 4 Comparison results of SABB and BRBB algorithms for solving P

(m, n, p)	Opt.val		CPU				Iter			
	SABB	BRBB	SABB	BRBB	Gap	Gap(%)	SABB	BRBB	Gap	Gap(%)
(500, 10, 2)	2.1055	2.1055	**0.10**	0.22	0.11	51.55%	**6.4**	15.2	8.8	57.9%
(500, 50, 2)	− 15.1191	− 15.1191	**0.41**	0.61	0.19	31.97%	**12.6**	22.3	9.7	43.5%
(500, 100, 2)	10.3810	10.3810	**0.73**	1.15	0.42	36.40%	**13.0**	22.3	9.3	41.7%
(500, 200, 2)	− 3.9477	− 3.9477	**2.15**	3.83	1.68	43.97%	**13.5**	27.0	13.5	50.2%
(500, 500, 2)	− 25.2257	− 25.2257	**18.85**	27.17	8.33	30.64%	**19.5**	30.0	10.5	35.0%
(500, 1000, 2)	− 9.3850	− 9.3850	**86.13**	111.90	25.7	23.03%	**26.3**	37.9	11.6	30.6%
(500, 2000, 2)	− 3.7374	− 3.7374	**146.09**	236.39	90.30	38.20%	**12.4**	24.8	12.4	50.0%
(500, 5000, 2)	− 11.6333	− 11.6333	**2171.26**	2258.17	86.91	4.00%	**25.5**	31.5	6.0	19.1%
(500, 6500, 2)	− 36.0924	..	**2520.72**	..	..	..	**14**	..	..	..
(500, 50, 5)	− 99.6634	− 99.6634	**1.90**	4.22	2.32	54.93%	**69.6**	167.2	97.6	58.4%
(500, 100, 5)	7.6595	7.6595	**5.00**	8.93	3.93	44.04%	**81.8**	162.4	80.7	49.7%
(500, 200, 5)	− 21.3063	− 21.3063	**36.29**	62.26	25.96	41.70%	**217.5**	424.9	207.4	48.8%
(500, 500, 5)	− 25.6952	− 25.6952	**466.93**	560.67	93.74	16.72%	**476.0**	641.8	165.8	25.8%
(500, 1000, 5)	− 2.0086	− 2.0086	**1016.21**	1507.35	491.14	32.58%	**273.7**	459.7	186.0	40.5%
(500, 2000, 5)	54.1313	54.1313	**2312.25**	2615.09	302.84	11.58%	**179**	227	48	21.2%
(500, 3000, 5)	27.6789	..	**2916.85**	..	..	..	**774.5**	..	..	..

Table 5 Comparison results of ARBB and CRBB algorithms for solving P

(m, n, p)	CPU					Iter				
	SABB	CRBB				SABB	CRBB			
		$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0.7$	$\alpha = 1.0$		$\alpha = 0.3$	$\alpha = 0.5$	$\alpha = 0.7$	$\alpha = 1.0$
(100, 10, 2)	0.22	0.23	0.22	0.19	**0.17**	10.7	11.3	10.7	9.0	**7.7**
(100, 50, 2)	0.46	0.43	0.44	0.42	**0.34**	19.5	18.3	18.3	17.6	**13.7**
(100, 100, 2)	0.54	0.64	0.61	0.52	**0.42**	15.6	19.1	18.2	15.2	**11.5**
(100, 200, 2)	**2.69**	2.92	2.71	2.75	2.97	**47.9**	52.3	48.4	48.9	52.5
(100, 300, 2)	**3.06**	3.65	3.35	3.38	3.51	**30.3**	36.5	33.5	33.5	34.7
(100, 400, 2)	**4.22**	4.58	4.54	4.42	4.75	**25.4**	27.8	27.7	26.9	28.6
(100, 500, 2)	**11.84**	13.08	12.71	12.64	13.87	**48.6**	53.8	52.3	52.1	56.7
(100, 600, 2)	**10.18**	12.41	10.85	10.36	11.08	29.3	36.1	31.5	30.2	**29.1**
(100, 700, 2)	**18.22**	22.11	20.17	21.21	20.19	**39.7**	48.5	44.3	46.4	43.9
(100, 800, 2)	**12.04**	14.27	14.13	13.11	12.44	20.3	24.2	24.2	22.3	**19.4**
(100, 1000, 2)	**35.67**	41.08	42.32	43.03	40.51	**37.0**	42.6	43.9	44.7	42.1
(100, 2000, 2)	**186.18**	203.05	206.06	211.83	239.35	**46.7**	51.2	52.1	53.3	60.0
(100, 5000, 2)	**1450.72**	1623.30	1658.99	1686.74	1677.31	**45.3**	51.1	52.5	53.5	52.9
(100, 10, 5)	1.05	1.27	1.14	1.03	**0.98**	**34.9**	48.9	41.9	37.5	35.9
(100, 50, 5)	6.11	7.43	6.96	6.62	**5.49**	**151.6**	220.2	206.0	179.0	162.7
(100, 100, 5)	**10.39**	16.07	15.67	15.14	12.36	**254.9**	393.4	380.2	364.1	289.2
(100, 500, 5)	**41.51**	56.99	54.12	56.10	52.14	**493.9**	701.3	755.0	729.3	574.1
(100, 1000, 5)	**362.63**	528.38	504.19	474.52	449.14	**984.2**	1474.1	1393.0	1304.8	1220.9
(100, 2000, 5)	**718.35**	935.69	885.18	841.41	799.18	**1080.1**	1534.8	1460.8	1370.9	1276.0
(100, 5000, 5)	**1863.50**	2208.85	2179.41	2186.37	2148.25	**1763.0**	2121.3	2084.6	2095.3	030.9

5.3 Numerical Comparison of the Branching Direction

In this subsection, we test randomly generated problem P to demonstrate the impact of the chosen direction for self-adjustable branching rule on the algorithm.

Table 6 shows that both SABB algorithm and SABB-L algorithm can find the same optimal values for problem P when $(m, p) = (50, 2)$. However, the CPU time cost by SABB algorithm is reduced by at lest 57.18% compared to SABB-L algorithm. And the number of iterations required by SABB algorithm is significantly lower than that by SABB-L algorithm. Note that SABB algorithm terminates and returns the optimal solution when p takes the values of $3, 4$ and 5, while SABB-L algorithm is not terminated and keeping run after obtaining the same optimal values. These results indicate that the choosing branching direction of SABB algorithm has higher computing efficiency comparing with the general way of choosing directions.

5.4 Numerical Comparison of Algorithms

In this subsection, the comparisons of SABB algorithm and the algorithms in [29, 32] for solving P are performed.

As shown in Table 7, SABB algorithm and the algorithm in [32] can find the same optimal values for all test random instances, whereas the CPU time cost by SABB algorithm is less than those of the algorithm in [32]. Moreover, the CPU time spent by SABB algorithm grows slowly with the increase of n compared to the other two algorithms. By contrast, the algorithm in [29] finds the global optimal values for only a few random instances, and it costs longer the CPU time than the other two algorithms. These results reveal that the self-adjustable branching rule is helpful to improve the efficiency of SABB algorithm.

6 Conclusions

In this paper, we investigate a linear multiplicative program (LMP), which has important application in various domains such as financial optimization and robust optimization. Firstly, by employing appropriate variable substitution techniques, LMP is transformed into an equivalent problem (EP). Subsequently, EP can be simplified into a series of linear relaxation programs through the application of affine approximations. Then we propose a self-adjustable branch-and-bound algorithm by integrating the self-adjustable branching rule and branch-and-bound framework. The proposed algorithm has been proven to converge to the global optimal solution of the initial LMP. Additionally, we conduct an analysis of the computational complexity of the algorithm. Finally, numerical results demonstrate its feasibility and high efficiency. The future research direction is to investigate whether there exist more effective relaxation methods, alternative branching rules or reduction strategies for addressing general linear multiplicative problem.

Table 6 Comparison results of the branching direction for solving P

(m, n, p)	Opt.val			CPU				Iter			
	SABB	SABB-L	Gap	SABB	SABB-L	Gap	Gap (%)	SABB	SABB-L	Gap	Gap (%)
(50, 5, 2)	−0.0044	−0.0044	0.0000	**0.10**	0.24	0.14	59.53	**1.7**	9.4	7.7	81.82
(50, 15, 2)	0.0105	0.0105	0.0000	**0.11**	0.35	0.24	67.80	**2.6**	15.6	13.0	83.57
(50, 50, 2)	−0.0263	−0.0263	0.0000	**0.22**	0.60	0.39	64.20	**6.5**	23.0	16.5	71.74
(50, 100, 2)	0.0227	0.0227	0.0000	**0.27**	5.66	5.39	95.15	**6.2**	159.2	153.0	96.11
(50, 200, 2)	0.0513	0.0513	0.0000	**0.50**	7.09	6.58	92.88	**7.2**	116.1	108.9	93.78
(50, 500, 2)	0.0895	0.0894	0.0000	**3.81**	8.89	5.08	57.18	**15.2**	36.2	21.0	58.01
(50, 5, 3)	0.0257	0.0257	0.0000	**0.15**	3.11	2.97	95.42	**2.8**	159.5	156.7	98.21
(50, 10, 3)	−0.0638	−0.0638	0.0000	**0.12**	4.07	3.96	97.20	**3.0**	209.6	206.6	98.57
(50, 50, 3)	0.0961	0.0961	0.0000	**0.37**	–	9.64	–	**11.3**	398.6	387.3	–
(50, 100, 3)	0.0336	0.0336	0.0000	**0.91**	–	9.11	–	**22.8**	276.1	253.3	–
(50, 200, 3)	**0.0725**	0.0721	0.0003	**3.03**	–	7.02	–	**46.7**	157.7	111.0	–
(50, 500, 3)	**0.0888**	0.0884	0.0004	**6.72**	–	3.42	–	**25.7**	39.9	14.2	–
(50, 5, 4)	0.0117	0.0117	0.0000	**0.24**	5.59	5.35	95.68	**5.8**	283.0	277.2	98.57
(50, 10, 4)	0.0408	0.0408	0.0000	**0.40**	–	9.62	–	**14.0**	497.6	483.6	–
(50, 50, 4)	**−0.0646**	−0.0647	0.0001	**0.81**	–	9.21	–	**27.3**	394.6	367.3	–
(50, 100, 4)	0.0496	0.0496	0.0000	**1.63**	–	8.38	–	**39.5**	270.2	230.7	–

Table 6 continued

(m, n, p)	Opt.val			CPU				Iter			
	SABB	SABB-L	Gap	SABB	SABB-L	Gap	Gap (%)	SABB	SABB-L	Gap	Gap (%)
(50, 200, 4)	0.0570	0.0570	0.0000	**2.64**	–	7.39	–	**38.9**	154.1	115.2	–
(50, 500, 4)	**0.1508**	0.1507	0.0001	**7.31**	–	2.81	–	**26.8**	37.3	10.5	–
(50, 5, 5)	−0.0426	−0.0426	0.0000	**0.37**	8,38	8.00	95.52	**9.8**	410.5	400.8	97.62
(50, 10, 5)	0.0655	0.0655	0.0000	**0.67**	–	9.34	–	**25.9**	495.2	469.3	–
(50, 50, 5)	**0.0099**	0.0098	0.0001	**2.15**	–	7.86	–	**75.3**	379.4	304.1	–
(50, 100, 5)	0.0840	0.0840	0.0000	**2.77**	–	7.25	–	**68.3**	264.3	196.0	–
(50, 200, 5)	0.1759	0.1759	0.0000	**3.19**	–	6.83	–	**42.9**	141.0	98.1	–
(50, 500, 5)	0.1217	0.1217	0.0000	**9.24**	–	0.36	–	**46.5**	178.0	131.5	–

Table 7 Comparison results of SABB algorithm and the algorithms in [29, 32] for solving P

(m, n, p)	Opt. val			CPU			Iter		
	SABB	[32]	[29]	SABB	[32]	[29]	SABB	[32]	[29]
(50, 10, 2)	10.0331	10.0331	10.0331	**0.36**	1.87	6.35	56.8	298.8	133.2
(50, 50, 2)	4.0225	4.0225	4.0225	**0.53**	1.65	115.86	65.0	199.2	2272.2
(50, 100, 2)	8.8669	8.8669	8.8669	**0.86**	1.00	659.14	64.0	73.0	10926.0
(50, 200, 2)	5.1865	5.1865	5.1865	**6.54**	13.21	2509.93	291.3	561.2	21716.0
(50, 500, 2)	19.5145	19.5145	..	**16.61**	53.57	..	180.4	538.1	..
(50, 1000, 2)	22.8407	22.8407	..	**190.55**	486.49	..	532.0	1391.0	..
(50, 2000, 2)	14.9818	14.9818	..	**182.05**	763.55	..	171.8	558.0	..
(50, 3000, 2)	19.5314	19.5314	..	**463.65**	1786.91	..	121.4	401.0	..
(50, 4000, 2)	19.3124	19.3124	..	**783.03**	2315.54	..	158.3	439.5	..
(50, 5000, 2)	15.6204	15.6204	..	**1687.20**	2835.89	..	132.7	337.0	..
(50, 6000, 2)	14.8885	..	..	**1936.91**	..	..	105.5	..	..
(50, 10, 5)	5.7677	5.7677	5.7677	**1.75**	2.90	29.21	73.0	136.6	81.8
(50, 50, 5)	−4.0862	−4.0862	−4.0862	**44.36**	63.63	1561.28	1800.0	2544.0	4138.0
(50, 100, 5)	24.2594	24.2594	..	**72.52**	245.05	..	1820	5784	..
(50, 400, 5)	−17.7876	..	..	**3183.39**	..	..	36,552	..	..

Acknowledgements The authors are grateful to the responsible editor and the anonymous referees for their valuable comments and suggestions, which has helped to substantially improve the presentation of this work.

Author Contributions The whole work has been carried out by the author.

Funding Not applicable.

Availability of Data and Materials Not applicable.

Declarations

Conflict of interest No potential conflict of interest was reported by the author.

Ethics Approval Not applicable.

References

1. Kahl, F., Agarwal, S., Chandraker, M.K., Kriegman, D., Belongies, S.: Practical global optimization for multiview geometry. Int. J. Comput. Vis. **79**(3), 271–284 (2008)
2. Qu, S., Zhou, Y., Zhang, Y., Wahab, M.I.M., Zhang, G., Ye, Y.: Optimal strategy for a green supply chain considering shipping policy and default risk. Comput. Ind. Eng. **131**, 172–186 (2019)
3. Konno, H., Shirakawa, H., Yamazaki, H.: A mean-absolute deviation-skewness portfolio optimization model. Ann. Oper. Res. **45**, 205–220 (1993)
4. Konno, H., Kuno, T.: Generalized linear multiplicative and fractional programming. Ann. Oper. Res. **25**, 147–162 (1990)
5. Quesada, I., Grossmann, I.E.: Alternative bounding applications for the global optimization of various engineering design problems. In: Grossmann, I.E. (ed.) Global Optimization in Engineering Design. Nonconvex Optimization and Its Applications, vol. 9, pp. 309–331. Springer, Berlin (1996)
6. Bennett, K., Mangasarian, O.: Bilinear separation of two sets in n-space. Comput. Optim. Appl. **2**, 207–227 (1994)
7. Dorneich, M., Sahinidis, N.: Global optimization algorithms for chip design and compaction. Eng. Optim. **25**(2), 131–154 (1995)
8. Mulvey, J., Vanderbei, R., Zenios, S.: Robust optimization of large-scale systems. Oper. Res. **43**, 264–281 (1995)
9. Tuy, H.: Convex Analysis and Global Optimization, 2nd edn. Kluwer Academic, Dordrecht (2016)
10. Benson, H.: Global maximization of a generalized concave multiplicative function. J. Optim. Theory Appl. **137**, 105–120 (2008)
11. Zhao, Y., Liu, S.: Global optimization algorithm for mixed integer quadratically constrained quadratic program. J. Comput. Appl. Math. **319**, 159–169 (2017)
12. Lu, C., Deng, Z., Jin, Q.: An eigenvalue decomposition based branch-and-bound algorithm for non-convex quadratic programming problems with convex quadratic constraints. J. Global Optim. **67**(3), 475–493 (2017)
13. Luo, H., Chen, S., Wu, H.: A new branch-and-cut algorithm for non-convex quadratic programming via alternative direction method and semidefinite relaxation. Numer. Algorithms **88**, 993–1024 (2021)
14. Konno, H., Kuno, T., Yajima, Y.: Parametric simplex algorithms for a class of NP-complete problems whose average number of steps is polynomial. Comput. Optim. Appl. **1**, 227–239 (1992)
15. Raghavachari, M.: On connections between zero-one integer programming and concave programming under linear constraints. Oper. Res. **17**, 680–684 (1969)
16. Matsui, T.: NP-hardness of linear multiplicative programming and related problems. J. Global Optim. **9**(2), 113–119 (1996)
17. Konno, H., Kuno, T.: Linear multiplicative programming. Math. Program. **56**, 51–64 (1992)
18. Ryoo, H.S., Sahinidis, N.V.: Global optimization of multiplicative programs. J. Global Optim. **26**, 387–418 (2003)

19. Gao, Y., Xu, C., Yang, Y.: Outcome-space branch and bound algorithm for solving linear multiplicative programming. Comput. Intell. Secur. **3801**, 675–681 (2005)
20. Zhou, X., Cao, B., Wu, K.: Global optimization method for linear multiplicative programming. Acta Math. Appl. Sin. **31**(2), 325–334 (2015)
21. Cambini, R., Riccardi, R., Scopelliti, D.: Solving linear multiplicative programs via branch-and-bound: a computational experience. CMS **20**(1), 38 (2023)
22. Cambini, R., Sodini, C.: Global optimization of a rank-two nonconvex program. Math. Methods Oper. Res. **71**(1), 165–180 (2010)
23. Cambini, R., Sodini, C.: On the minimization of a class of generalized linear functions on a flow polytope. Optimization **63**(10), 1449–1464 (2014)
24. Yang, L., Shen, P., Pei, Y.: A global optimization approach for solving generalized nonlinear multiplicative programming problem. Abstr. Appl. Anal. **2014**(1), 641909 (2014)
25. Gao, Y., Xu, C., Yang, Y.: An outcome-space finite algorithm for solving linear multiplicative programming. Appl. Math. Comput. **179**(2), 494–505 (2006)
26. Oliveira, Rúbia. M., Ferreira, P.A.V.: An outcome space approach for generalized convex multiplicative programs. J. Global Optim. **47**(1), 107–118 (2010)
27. Shen, P., Huang, B., Wang, L.: Range division and linearization algorithm for a class of linear ratios optimization problems. J. Comput. Appl. Math. **350**, 324–342 (2019)
28. Liu, S., Zhao, Y.: An efficient algorithm for globally solving generalized linear multiplicative programming. J. Comput. Appl. Math. **296**, 840–847 (2016)
29. Wang, C., Bai, Y., Shen, P.: A practicable branch-and-bound algorithm for globally solving multiplicative programming. Optimization **66**(3), 397–405 (2017)
30. Wang, C., Deng, Y., Shen, P.: A novel convex relaxation-strategy-based algorithm for solving linear multiplicative problems. J. Comput. Appl. Math. **407**, 114080 (2022)
31. Zhao, Y., Zhao, T.: Global optimization for generalized linear multiplicative programming using convex relaxation. Math. Problems Eng. **2018**, 9146309 (2018)
32. Yin, J., Jiao, H., Shang, Y.: Global algorithm for generalized affine multiplicative programming Problem. IEEE Access **7**, 162245–162253 (2019)
33. Shen, P., Wang, K., Lu, T.: Outer space branch and bound algorithm for solving linear multiplicative programming problems. J. Global Optim. **78**, 453–482 (2020)
34. Shen, P., Huang, B.: Global algorithm for solving linear multiplicative programming problems. Optim. Lett. **14**, 693–710 (2020)
35. Shen, P., Wang, K., Lu, T.: Global optimization algorithm for solving linear multiplicative programming problems. Optimization **71**(6), 1421–1441 (2022)
36. Shen, P., Wu, D., Wang, F.: An efficient spatial branch-and-bound algorithm using an adaptive branching rule for linear multiplicative programming. J. Comput. Appl. Math. **426**, 115100 (2023)
37. Shen, P., Wu, D., Wang, K.: Globally minimizing a class of linear multiplicative forms via simplicial branch-and-bound. J. Global Optim. **86**, 303–321 (2023)
38. Jiao, H., Wang, W., Chen, R., et al.: An efficient outer space algorithm for generalized linear multiplicative programming problem. IEEE Access **99**, 1–1 (2020)
39. McCormick, G.P.: Computability of global solutions to factorable nonconvex programs: Part I-Convex underestimating problems. Math. Program. **10**(1), 147–175 (1976)