Research Article

Image Space Accelerating Algorithm for Solving a Class of Multiplicative Programming Problems

Haoyu Zhou 1, Guohou Li 2, Xueliang Gao 3, and Zhisong Hou 2,4

1School of Food Science, Henan Institute of Science and Technology, Xinxiang 453003, China
2School of Information Engineering, Henan Institute of Science and Technology, Xinxiang 453003, China
3School of Mathematical Sciences, Henan Institute of Science and Technology, Xinxiang 453003, China
4School of Computer Science and Technology, Xidian University, Xi’an 710071, China

Correspondence should be addressed to Zhisong Hou; houzhs@gmail.com

Received 7 January 2022; Revised 17 May 2022; Accepted 30 May 2022; Published 20 July 2022

Academic Editor: Federica Caselli

Copyright © 2022 Haoyu Zhou et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper interprets an image space accelerating branch and bound algorithm for globally solving a class of multiplicative programming problems (MP). In this algorithm, in order to obtain the global optimal solution, the problem (MP) is transformed into an equivalent problem (P2) by introducing new variables. By utilizing new linearizing relaxation technique, the problem (P2) can be converted into a series of linear relaxation programming problems, which provide the reliable lower bound in the branch and bound search. Meanwhile, an image space accelerating method is constructed to improve the computational performance of the algorithm by deleting the subintervals which have no global optimal solution. Furthermore, the global convergence of the algorithm is proved. The computational complexity of the algorithm is analyzed, and the maximum iterations of the algorithm are estimated. Finally, numerical experimental results show that the algorithm is robust and efficient.

1. Introduction

Consider a class of multiplicative programming problems as follows:

\[(MP) \begin{align*}
\min_{x} & \quad \phi(x) = \prod_{j=1}^{p} (c_j^T x + d_j)^{y_j}, \\
\text{s.t.} & \quad x \in X = \{ x | Ax \leq b \},
\end{align*}\]  

where \( p \in \mathbb{Z}^+; c_j \in \mathbb{R}^n, d_j, y_j \in \mathbb{R}, j = 1, 2, \ldots, p; A \in \mathbb{R}^{m \times n}, \) and \( b \in \mathbb{R}^m; c_j^T x + d_j > 0 \) for any \( x \in X; \) and the feasible field \( X \) is a nonempty bound polyhedron set. The problem is well known as multiplicative programming problem that corresponds to a nonlinear optimization problem with non-convex objective function.

The problem (MP) comes from various application fields, for example, decision tree optimization [1], multiple-objective decision [2, 3], robust optimization [4], control and engineering optimization [5–9], computer vision [10, 11], network flow optimization [12–14], food science engineering, and big data analysis. That is to say, the problem is useful in many aspects, so it is of great importance to develop an efficient algorithm for the problem (MP).

In the past twenty years, many deterministic feasible algorithms have been developed to solve the problem (MP). Generally, these algorithms can be divided into several categories, such as parameterization based methods [15–17], outer-approximation methods [18, 19], finite branch and bound algorithms [20–28], decomposition method [29], level set algorithms [30, 31], primal and dual simplex method [32], cutting plane methods [16, 33], and heuristic methods [34, 35]. Recently, Gao et al. [19] proposed an outcome-space finite algorithm to solve a class of linear multiplicative program problems by solving only a convex quadratic program problem in each iteration. Chen and Jiao [36] present a finite branch and bound algorithm based on the theory of monotonic optimization. Wang et al. [37] used simplicial subdivision in the branch operation to solve generalized multiplicative programs. By directly mapping the original variable space to two-dimensional space, Wang
et al. [38] gave a method that can effectively solve special second-order multiplicative problems. Shen and Huang [39] developed a rectangular branch and bound algorithm with bisection rule to solve linear multiplicative programming problems. Liu and Zhao [31] presented a new algorithm for solving a generalized multiplicative programming based on level set method. Zhang et al. [40] proposed an output space and bound algorithm for solving the linear multiplicative programs. Moreover, Shen and Wang [41] proposed a branch and bound algorithm that can solve the multiplicative programs in polynomial time. Using a piecewise linear approximation method and some outer space accelerating techniques, Hou and Liu [42] provided an outer space algorithm to solve a class of multiplicative programs effectively. Up to now, although there is a lot of progress in the development of deterministic algorithms for solving the special forms of the problem (MP), the global optimization algorithm of general forms of the problem (EP) in this paper has been little researched.

In this paper, based on the branch and bound framework, a new image space accelerating algorithm is proposed by solving a series of linear relaxation problems over partitioned subsets to find a global optimal solution of the problem (MP). For this purpose, we first convert the problem (MP) into the equivalent optimization problem (P2) by mapping the problem (MP) from the origin space to image space. Second, a linear relaxation technique of the problem (P2) is constructed. Based on the linear relaxation technique, the problem (P2) is systematically converted into a sequence of linear relaxation programming problems. Third, by successive refinement iterations, the solutions of these constructed linear relaxation problems can be as close as possible to the global optimal solution of the problem (MP). Meanwhile, in the above iterations, the approximation process can be accelerated by deleting some regions in which there is no global optimal solution of the problem (P2).

Compared with the known methods, the proposed algorithm has the following superiorities: (1) new variables are introduced to reduce n-dimensional space into p-dimensional image space. Then, the branching operations in branch and bound framework act on the p-dimensional image space which can significantly reduce the computational cost as p is usually smaller than n. (2) In the bound operations, according to properties of the objective function of the problem (P2) and the structure of the algorithm, an accelerating method is proposed to reduce the computational cost of the algorithm. (3) We prove the global convergence of the proposed algorithm and estimate the maximum number of the iterations by analyzing the computational complexity of the algorithm. On the other hand, the efficiency of the algorithm is sensitive to the interval range of the variable of the image space.

The remainder of this paper is organized as follows: In Section 2, we give the equivalent transformation of the origin problem and the linear relaxation method for the equivalent problem. In Section 3, an image space accelerating algorithm based on branch and bound framework is developed and the convergence of the algorithm is proved. In addition, the computational complexity of the algorithm is analyzed and the maximum number of iterations of the algorithm is estimated. In Section 4, numerical results of some tests are provided. Finally, a brief summary is made in Section 5.

2. Equivalent Problem and Its Linear Relaxation

In this section, we will convert the problem (MP) into an equivalent problem (P1) which has the same optimal solution as the problem (MP). Using the logarithmic function property, we can obtain this equivalent programming problem as follows:

\[
\text{(P1): } \begin{cases}
\min \sum_{j=1}^{p} y_j \ln (c_j^T x + d_j), \\
\text{s.t., } x \in X = \{ x | Ax \leq b \}.
\end{cases}
\]

Next, introduce \( p \) variables \( t_j = c_j^T x + d_j, j = 1, 2, \ldots, p \) and let \( l^0_j \) and \( u^0_j \) be the minimum and maximum of \( t_j, j = 1, 2, \ldots, p \), which can be obtained by the following linear programming:

\[
l^0_j = \min_{Ax \leq b} \{ c_j^T x + d_j \},
\]

and

\[
u^0_j = \max_{Ax \leq b} \{ c_j^T x + d_j \}.
\]

Hence, the problem (P1) can be transformed into the following equivalent problem (P2).

\[
\text{(P2): } \begin{cases}
\min \Psi(t) = \sum_{j=1}^{p} y_j \ln (t_j), \\
\text{s.t., } t_j = c_j^T x + d_j, j = 1, 2, \ldots, p, \\
t_j \in [l^0_j, u^0_j].
\end{cases}
\]

Obviously, the problem (P1) has the same optimal solution as the problem (P2). In the following, the main task is to solve the problem (P2). For solving the problem (P2), the main work is to construct lower bounds of this problem and its subproblems generated by the branch and bound algorithm. Meanwhile, the lower bounds of this problem and its subproblems can be obtained by solving the corresponding linear relaxation programming problems.

For any \( t \in [l^0_j, u^0_j] \), we define the following functions:

\[
k_j = \frac{\ln(u_j) - \ln(l_j)}{u_j - l_j},
\]

\[
h_j(t_j) = \ln(l_j) + k_j(t_j - l_j),
\]

\[
g_j(t_j) = k_j t_j - \ln(k_j) - 1,
\]

\[
f_j(t_j) = \ln(t_j).
\]
For any functions $f_j(t_j), g_j(t_j),$ and $h_j(t_j)$ over $[l_j, u_j], j = 1, 2, \ldots, p$ with a linear function. The following theorem will show the details of this linear relaxation technique for the problem (P2).

**Theorem 1.** For any functions $f_j(t_j), g_j(t_j),$ and $h_j(t_j)$ over $[l_j, u_j], j = 1, 2, \ldots, p,$ we have the following two statements:

1. The function $h_j(t_j)$ is the linear convex envelope of the function $f_j(t_j),$ and the function $g_j(t_j)$ is a supporting hyperplane of the function $f_j(t_j)$ which is parallel with $h_j(t_j).$ Meanwhile, we have

$$h_j(t_j) \leq f_j(t_j) \leq g_j(t_j),$$

for $\forall t_j \in [l_j, u_j].$ (7)

2. Define

$$\delta^j_1(t_j) = g_j(t_j) - f_j(t_j),$$

$$\delta^j_2(t_j) = f_j(t_j) - h_j(t_j),$$

then, we have

$$\max_{t_j \in [l_j, u_j]} \delta^j_1(t_j) \to 0,$$

$$\max_{t_j \in [l_j, u_j]} \delta^j_2(t_j) \to 0$$

as $\omega_j \to 0.$ (9)

**Proof.** (i) Obviously, the function $f_j(t_j) = \ln(t_j)$ is a concave and monotone increase function of $t_j$ over $[l_j, u_j].$ Therefore, the line passing through the points $(l_j, \ln(l_j))$ and $(u_j, \ln(u_j))$ is the affine convex envelope of the function $f_j(t_j) = \ln(t_j),$ and this line can be described as the form of $h_j(t) = \ln(t_j) + k_j(t_j - l_j).$ Hence, by $t_j \in [l_j, u_j],$ we have

$$h_j(t_j) = \ln(l_j) + k_j(t_j - l_j) \leq \ln(t_j) = f_j(t_j).$$

On the other hand, suppose that the line $g_j(t_j)$ is the tangential plane of the function $f_j(t)$ and parallel to the line $h_j(t_j),$ the tangential support point must appear at the point $(1/k_j, \ln(1/k_j)).$ Then, the corresponding tangential function of this line is $g_j(t) = k_j t_j - \ln(k_j) - 1.$ According to the geometric properties of the function $f_j(t_j) = \ln(t_j)$ for $\forall t_j \in [l_j, u_j],$ we obtain

$$f_j(t_j) = \ln(t_j) \leq k_j t_j - \ln(k_j) - 1 = g_j(t_j).$$

This completes the proof of (i) in Theorem 1. With $z_j = u_j/l_j,$ $k_j = \ln(u_j) - \ln(l_j)/u_j - l_j,$ $k_j$ can be changed into

$$k_j = \frac{\ln(z_j)}{l_j(z_j - 1)}.$$ (12)

Then, we can obtain

$$\delta^j_1(t_j) = k_j t_j - 1 - \ln(k_j) - \ln(t_j),$$

$$\delta^j_2(t_j) = \ln(t_j) - \ln(t_j) - k_j(t_j - l_j).$$

It is known that $\delta^j_2(t_j)$ is a concave function of $t_j$ over $[l_j, u_j],$ so the maximum $\delta^j_2_{\text{max}}$ can be obtained at the point of tangential support whose value is $t_j = 1/k_j.$ Thus,

$$\delta^j_2_{\text{max}} = \ln\left(\frac{1}{k_j}\right) - \ln(t_j) - k_j\left(\frac{1}{k_j} - t_j\right)$$

$$= -\ln(k_j) - 1 + k_j t_j$$

$$= \ln\left(\frac{z_j - 1}{\ln(z_j)}\right) + 1 + \frac{\ln(z_j)}{z_j - 1}.$$

From $z_j \to 1,$ $\ln(z_j) - 1 \to 0$ as $\omega_j \to 0,$ we can obtain that $\delta^j_2_{\text{max}} \to 0$ as $\omega_j \to 0.$

On the other hand, $\delta^j_1(t_j)$ is a convex function of $t_j$ for any $t_j \in [l_j, u_j].$ Clearly, the maximum of $\delta^j_1(t_j),$ named $\delta^j_{1,\text{max}}$ must occur at the critical point, $l_j$ or $u_j.$ In addition, the form of $k_j = \ln(u_j) - \ln(l_j)/u_j - l_j$ can be transformed into the form of

$$k_j u_j - \ln(u_j) = k_j l_j - \ln(l_j).$$

Then, we have

$$\delta^j_1(t_j) = k_j l_j - 1 - \ln(k_j) - \ln(t_j)$$

$$= \delta^j_1_{\text{max}},$$

$$\delta^j_1(u_j) = k_j u_j - 1 - \ln(k_j) - \ln(u_j)$$

$$= k_j l_j - \ln(t_j) - \ln(k_j) - 1$$

$$= \delta^j_1(t_j)$$

$$= \delta^j_1_{\text{max}}.$$ (17)

From the description shown above, it holds that $\delta^j_{1,\text{max}} = \delta^j_1(l_j) = \delta^j_1(u_j) = \delta^j_{1,\text{max}}$ and $\delta^j_{1,\text{max}} \to 0, \delta^j_2_{\text{max}} \to 0$ as $\omega_j \to 0.$ This completes the proof of (ii). \[\square\]

**Remark.** From Theorem 1, we can get approximations solution by shortening the gap between tangential hypersurface $g_j(t)$ and concave envelope $h_j(t).$ As a result, the functions $g_j(t)$ and $h_j(t)$ infinitely approximate the function $f_j(t)$ when $\omega_j \to 0.$

Now, we give the linear relaxation programming of the problem (P2). Suppose that $T^k = [l^k, u^k] \subseteq [l^0, u^0].$ Denoting that $\Psi^k_j(t_j)$ is the underestimating function for each nonlinear function $\gamma_j(t_j)$ over $T^k_j = [l^k_j, u^k_j] \subseteq [l_j, u_j],$ for any $t_j \in [l_j, u_j],$ we obtain
programming problem, all feasible solutions of the problem are constructed as follows:

\[ y_j \ln (t_j) \geq \Psi^k_j (t_j) \]

\[
= \begin{cases} 
  y_jh_j(t_j) & \text{if } y_j > 0, \\
  y_jg_j(t_j) & \text{if } y_j < 0, 
\end{cases} 
\quad j = 1, 2, \ldots, p. \tag{18}
\]

Then, summing all the terms \( \Psi^L_j (t_j), j = 1, 2, \ldots, p \) and expressing the result as \( \Psi^R (t) = \sum_{j=1}^p \Psi^R_j (t_j) \), we have

\[ \Psi (t) \geq \Psi^R (t), \forall t \in [l_j, u_j]. \tag{19} \]

Consequently, the corresponding linear relaxation programming problem over \( T \) of the problem (P2) can be constructed as follows:

\[
(P3): \begin{aligned}
\min \quad & \Psi^R (t), \\
\text{s.t.} \quad & t_j = c_j x + d_j, \quad j = 1, 2, \ldots, p, \\
& t \in T = [l^0, u^0], \\
& x \in X = \{ x | Ax \leq b \}.
\end{aligned} \tag{20}
\]

From the construction process of the linear relaxation programming problem, all feasible solutions of the problem (P2) over subdomain \( T^k \) are feasible to the problem (P3). Furthermore, in the subdomain \( T^k \), the optimal value of the problem (P3) is less than that of the problem (P2). Therefore, the problem (P3) provides an effective lower bound for the solution of the problem (P2) in the partition set \( T^k \). It should be pointed out that the problem (P3) only contains the necessary constraints to ensure the global convergence of the algorithm.

3. Algorithm, Convergence, and Its Complexity

By combining the branching operation, the bound operation, and the accelerating method, we design an image space accelerating algorithm to solve the equivalent problem (P2). By solving a series of linear relaxation programming problems over the subset of \( T^0 \), this method finds a global optimal solution.

The branch operation of the branch and bound method divides the set \( T^0 \) into subhyperrectangles. Each subhyperrectangle is related to a node of the branch and bound tree, and each node is associated with the linear relaxation programming subproblem in the corresponding subhyperrectangle.

Using \( Q_k \) to represent the active node of the step \( k \) of the algorithm, we will get a collection of active nodes which associate with a subhyperrectangle \( T \subseteq T^0, \forall T \in Q_k \). By calculating the solution \( LB(T) \) of the problem (P3), we get the lower bound of the optimal value of the problem (P2) on each node \( Q_k \) at step \( k \). Thus, we denote \( LB_k = \min \{ LB(T) \}, \forall T \in Q_k \) by the lower bound of the optimal value of the problem (P2) over the entire initial box region \( T^0 \) at step \( k \).

If the solution of the problem (P3) is feasible for the problem (P2), the upper bound UB of the incumbent solution is updated. Then, for any step \( k \), the active node \( Q_k \) is satisfied with \( LB(T) < UB, \forall T \in Q_k \).

Next, an active node with the smallest lower bound is selected and its associated subhyperrectangle is divided into two subhyperrectangles. As mentioned above, the lower bound of each new node is calculated and a new set of active nodes for the next step are generated. The whole process is iterated until the global convergence is obtained.

The key factor to ensure the convergence to the global minimum is to select the appropriate partition strategy. In this paper, we choose a standard and simple bisection rule. This method makes all intervals of every variables tend to zero, so it is sufficient to guarantee the global convergence of the proposed algorithm. The rule of the branching operation is as follows.

Assuming that the current active node is the subhyperrectangle \( Y' = [y', y'] \subseteq Y^0 \), the branching variable \( y_s \) satisfying \( s = \arg \max \{ y_j', y_j', i = 1, 2, \ldots, p \} \) is selected to apply the bisection rule. Then, we divide the interval \( [y_j', y_j'] \) into two equal intervals \( [y_j', (y_j' + y_j')/2] \) and \( [(y_j' + y_j')/2, y_j'] \).

3.1. Accelerating Method. According to the structure of the branch and bound framework, we propose an accelerating technique to enhance the convergence speed of the algorithm by deleting the whole region or a part of the region in which there is no global optimal solution.

Without losing generality, assume that the current subhyperrectangle is \( T^k \), where \( i^k \in T^k = [l^k, u^k] \subseteq T^0, j = 1, 2, \ldots, p \), and UB is the current upper bound of the problem (P1) which is obtained by the proposed algorithm. Denote

\[ \Omega^k = \sum_{j=1, y_j > 0}^p y_j f(l^k_j) + \sum_{j=1, y_j < 0}^p y_j f(u^k_j), \]

\[ \Lambda^k = \sum_{j=1, j \neq s, y_j > 0}^p y_j f(l^k_j) + \sum_{j=1, j \neq s, y_j < 0}^p y_j f(u^k_j), \]

\[ \eta_s^k = \exp \left( \frac{UB - \lambda^k_s}{\gamma_s} \right), \quad s \in \{ 1, 2, \ldots, p \}, \]

\[ T_s^k = \begin{cases} 
  [l^k_s, u^k_s] & j = 1, 2, \ldots, p, j \neq s, \\
  [\eta_s^k l^k_s, u^k_s] & j = s, \quad \text{if } \eta_s^k l^k_s \leq u^k_s, \\
  [l^k_s, \eta_s^k u^k_s] & j = s, \quad \text{if } \eta_s^k u^k_s \leq l^k_s.
\end{cases} \]

The accelerating method can be given by Theorem 2.

**Theorem 2.** For any \( t^k \in T^k = \subseteq T^0, j = 1, 2, \ldots, p \), the following conclusions hold:

1. If \( \Omega^k > UB \), there exists no global optimal solution to the problem (P2) in the current subhyperrectangle \( T^k \).
(2) If $\Omega^k \leq UB$, it can be divided into two kinds of cases: for any index $s \in \{0, 1, 2, \ldots, p\}$, if $y_s > 0$, there is no optimal solution in $T^k_s$; on the contrary, if $y_s < 0$, there is no optimal solution in $T^k_s$.

Proof. (i) If $\Omega^k > UB$, we have

$$\min_{t \in \mathcal{T}^k} \Psi(t) = \min_{t \in \mathcal{T}^k} \sum_{j=1}^{p} y_j \ln t_j^k$$

$$= \sum_{j=1, j \neq s}^{p} y_j \ln t_j^k + \sum_{j=1, j \neq s}^{p} y_j \ln u_j^k$$

$$= \Omega^k > UB.$$  \hspace{1cm} (22)

It is obvious that the problem (P2) has no global optimal solution over the current subhyperrectangle $T^k_r$. Therefore, the current subhyperrectangle $T^k$ can be deleted. (ii) If $\Omega^k \leq UB$, for any $t^k \in T^k$, consider the $s_{th}$ dimensional variable $t_s^k$ of $t^k$; we have

$$y_s \ln \left( t_s^k \right) > UB - \Lambda_k,$$  \hspace{1cm} (23)

where $y_j > 0$. With

$$A^k = \sum_{j=1, j \neq s}^{p} y_j f\left( t_j^k \right) + \sum_{j=1, j \neq s}^{p} y_j f\left( u_j^k \right), \quad y_s > 0,$$  \hspace{1cm} (24)

we have

$$\min_{t \in \mathcal{T}^k} \Psi(t) = \min_{t \in \mathcal{T}^k} \sum_{j=1, j \neq s}^{p} y_j \ln t_j^k > \sum_{j=1, j \neq s}^{p} y_j \ln t_j^k$$

$$+ \sum_{j=1, j \neq s}^{p} y_j \ln u_j^k + y_s \ln \left( t_s^k \right) \hspace{1cm} (25)$$

$$= \Lambda_k + UB - \Lambda_k = UB.$$  \hspace{1cm} (25)

Therefore, the current subhyperrectangle $T^k_r$ does not contain the global optimal solution of the problem (P2). For any $t^k \in T^k_r$, consider the $s_{th}$ dimensional variable $t_s^k$ of $t^k$; we obtain

$$y_s \ln \left( t_s^k \right) > UB - \Lambda_k,$$  \hspace{1cm} (26)

where $y_j < 0$. With

$$A^k = \sum_{j=1, j \neq s}^{p} y_j f\left( t_j^k \right) + \sum_{j=1, j \neq s}^{p} y_j f\left( u_j^k \right), \quad y_s < 0,$$  \hspace{1cm} (27)

we have

$$\min_{t \in \mathcal{T}^k} \Psi(t) = \min_{t \in \mathcal{T}^k} \sum_{j=1, j \neq s}^{p} y_j \ln t_j^k > \sum_{j=1, j \neq s}^{p} y_j \ln t_j^k$$

$$+ \sum_{j=1, j \neq s}^{p} y_j \ln u_j^k + y_s \ln \left( u_s^k \right) \hspace{1cm} (28)$$

$$= \Lambda_k + UB - \Lambda_k = UB.$$  \hspace{1cm} (28)

Consequently, there is no global optimal solution over $T^k_r$. This completes the proof of Theorem 2.

From Theorem 2, we can construct the accelerating method to delete the invalid subhyperrectangle or reduce the ineffective interval of all dimensions over the hyperrectangle which does not contain the global optimal solution to the problem (P2). Consequently, the efficiency of the proposed algorithm is improved.

3.2. Algorithmic Statement. Denote $LB(T^k)$ by lower bound function of the problem (P2) over every subhyperrectangle $T^k$ and denote $t^k = t(T^k)$ by the element of corresponding argmin method. The specific steps of the proposed global optimization algorithm are as follows:

Step 0. Initialization:

(1) Initialize the iteration number $k = 0$, the set of all active node $Q = \varnothing$, the upper bound $UB = +\infty$, the lower bound $LB = -\infty$, the small enough accuracy tolerance $\varepsilon > 0$, and the set of feasible solution $F = \varnothing$.

(2) Define $l^0_j = \min t_j = c^*_j x + d^*_j$, and $u^0_j = \max t_j = c^*_j x + d^*_j$, $j = 1, 2, \ldots, p$. The upper bound $u^0_j$ and lower bound $l^0_j$ of $t_j$ are generated, and the initial interval $T^0$ of $t$ is obtained. Then, add the rectangle $T^0 = [P^0, u^0]$ to the activate node set $Q$.

Step 1. Solving the problem (P3) over the hyperrectangle $T = T^0$, the feasible solution $F = [t^0, x^0]$, and the optimal solution $LB = LB(T)$ is obtained. Update the initial upper bound $UB$ through substituting $t^0$ into the problem (P2). If $UB - LB \leq \varepsilon$, go to Step 6. Otherwise, proceed to Step 2.

Step 2. Select a branching variable $t_k$ to divide $T^k$ into two new subhyperrectangles according to the branching rule shown above and denote the set of new partition rectangles by $\mathcal{T}^k$.

(1) For each $T \in \mathcal{T}^k$, using the above accelerating technique to compress its range, still denote the remaining subrectangle by $T^k$.

(2) For each $T \in \mathcal{T}^k \neq \varnothing$, compute the lower bound $LB(T^k)$ and the feasible solution $[t^k, x^k]$ by solving linear relaxation programming problem (P3). If $UB < LB_k$, discard the corresponding subhyperrectangle $T$ from $T^k$; this is to say, $\mathcal{T}^k = \mathcal{T}^k - T$; and continue to process the next element of $\mathcal{T}^k$. Otherwise, calculate $UB^k$ and add the subhyperrectangle into the active node set $Q$. If $UB^k - UB < \varepsilon$, update the upper bound $UB = UB^k$ and the feasible solution $F = [t^k, x^k]$ similar as Step 1; then go to Step 6.

Step 3. The remaining active node set is $Q = (Q - \mathcal{T}^k) \cup \mathcal{T}^k$. If $Q = \varnothing$, go to Step 6.
Step 4. Update the lower bound $LB: = \inf \{LB(T) : UB - LB < \varepsilon\}$. If $UB - LB < \varepsilon$, go to Step 6. Otherwise, $k: = k + 1$ and move on to the next step.

Step 5. Choose an active node $T^k$ where $T^k = \arg\min_{T \in Q} LB(T)$, $t^k = t(T^k)$, and return to Step 2. Otherwise, move on to the next step.

Step 6. Stop the algorithm with $\exp(UB)$ as the optimal solution to the problem (MP) and the vector $x^*$ intercepted from $F$ as the feasible solution.

3.3. Convergence of the Algorithm. In this subsection, the global convergence properties of the proposed algorithm are given. It is obvious that the partitioning sets used by the proposed algorithm are all rectangular and compact. Tuy [43] pointed out that rectangular subdivisions are exhaustive. Therefore, the proposed algorithm is exhaustive according to the branch operation rule mentioned above. Otherwise, if the proposed algorithm is infinite, the subdivision of $T$ generates an infinite, nested sequence of partition sets $\{T^k\}$, and an associated sequence of diameters $d(T^k)$ satisfies the condition as follows:

\[
\lim_{k \to \infty} d(T^k) = 0, \quad \lim_{k \to \infty} T^k = \cap_{k} T^k = \bar{T}. \tag{29}
\]

Lemma 1. Given a function $\Psi(t)$ in (4), $\Psi^R(t)$ given by (19) is strongly consistent over $T^0$. That is, there are subsequences $\{T^q\}$ of subhyperrectangle $\{T^k\}$ and a corresponding sequence $\{r^q\}$ with $r^q \in T^k$ satisfying $\Psi^R(r^q) \to \Psi(\bar{T})$ as $q \to \infty$.

**Proof.** By exhaustive partitioning according to the branching rule, a sequence of subhyperrectangle $\{T^k\}$ that is exhaustive on $T^0$ is obtained. Meanwhile, a corresponding sequence $\{r^q\}$ in $T^k$ is obtained. Through an exhaustive subdivision, we get that $T^k = [\bar{T}]$ and $t^k = \bar{T}$. Since the upper bound and the lower bound over $T^k$ are both sequences in a compact space, there exists a convergent subsequence. Hence, $T^q = [r^q, \bar{T}]$ as $q \to \infty$. Corresponding to this subsequence, there is the sequence $\{\bar{T}, t^q, r^q\} \to (\bar{T}, t^k, \bar{T})$ as $q \to \infty$ with $r^q \in T^q$. Furthermore, according to (5), (18), (19), and (20), it follows that

\[
\Psi(t) = \sum_{j=1}^{p} \gamma_j f_j(t_j) = \sum_{j=1, \gamma_j > 0}^{k} \gamma_j f_j(t_j) + \sum_{j=1, \gamma_j > 0}^{k} \gamma_j f_j(t_j), \tag{30}
\]

\[
\Psi^R(t) = \sum_{j=1}^{p} \Psi^R(t_j) = \sum_{j=1, \gamma_j > 0}^{k} \gamma_j h_j(t_j) + \sum_{j=1, \gamma_j > 0}^{k} \gamma_j g_j(t_j). \tag{31}
\]

From Theorem 1, we have

\[
0 \leq g_j(t) - f_j(t) \leq \frac{\ln(z_j)}{z_j - 1} - 1 \tag{32}
\]

\[
+ \ln \frac{z_j - 1}{\ln z_j} \quad \forall j = 1, 2, \ldots, p,
\]

where $z_j = u_j/l_j$ and $\omega_j = u_j - l_j = t_j^1 - t_j^0$, for $t \in T^q = [t^0, t^1]$. Since $g_j(t^0_j)$ and $f_j(t^0_j)$ can be regarded as the continuous functions of $t^0_j$ and $\omega_j$ is the function of $(t^0_j, t^1_j)$, by the problem (P3), we have

\[
\lim_{q \to \infty} \left( g_j(t^0_j) - f_j(t^0_j) \right) = g_j(\bar{T}) - f_j(\bar{T}) = 0. \tag{33}
\]

Using the similar proof process to (32), we obtain

\[
\lim_{q \to \infty} \left( f_j(t^0_j) - h_j(t^0_j) \right) = f_j(\bar{T}) - h_j(\bar{T}) = 0. \tag{34}
\]

Therefore, it can be seen from (30), (31), (32), and (33) that

\[
\Psi^R(x^q) \to \Psi(\bar{T}) \quad \text{as} \quad q \to \infty, \tag{35}
\]

which satisfies the strongly consistent underestimating requirements.

Lemma 2. Given the function $\Psi(t)$ in (5), $\Psi(t)$ has tight bounds. That is, there exist the upper bound $\overline{\Psi}(t)$ and the lower bound $\underline{\Psi}(t)$ of $\Psi(t)$, respectively, such that, for any infinite sequence $T^k$ which is produced by the exhaustive subpartition with $T^k \to \bar{T}$, there exists $\lim_{k \to \infty} \underline{\Psi}(t^k) = \lim_{k \to \infty} \overline{\Psi}(t^k) = \Psi(\bar{T})$.

**Proof.** Just as Lemma 1, for any infinite subhyperrectangle sequence $\{T^k\}$, $T^k = [t^0, t^1]$, which is produced by an exhaustive subpartition of $T^0$ with $t^k \to \bar{T}$ as $k \to \infty$, there exists a sequence $\{t^0\}$ with $t^0 \in T^k$ such that $t^k \to \bar{T}$ as $k \to \infty$. Corresponding to this sequence, there exists the sequence $\{(t^0, t^1, t^0) \to (\bar{T}, t^k, \bar{T}) \to \bar{T} \to \bar{T} \to \cdots \}$ as $k \to \infty$. Corresponding to (30) and (31), define $\overline{\Psi}(t) = \Psi^R(x)$ and define the function as follows:

\[
\overline{\Psi}(t) = \sum_{j=1, \gamma_j > 0}^{p} \gamma_j g_j(t_j) + \sum_{j=1, \gamma_j > 0}^{p} \gamma_j h_j(t_j). \tag{36}
\]

By Theorem 1, it is obvious that

\[
\Psi(t) \leq \overline{\Psi}(t) \leq \underline{\Psi}(t), \quad \forall t \in T^k \tag{37}
\]

Denote $\bar{k}$, $l^k$ by the lower bound and upper bound for $T^k$, respectively. According to (ii) of Theorem 1, for any $t \in T^k$, we have
where \( \max_{t \in T} \delta_j^k(t) \) and \( \max_{t \in T} \delta_j^2(t) \) are given by Theorem 1. Since \( z_j \rightarrow 1 \) as \( k \rightarrow \infty \), thus, \( \max_{t \in T} \delta_j^k(t) \rightarrow 0 \) and \( \max_{t \in T} \delta_j^2(t) \rightarrow 0 \) as \( k \rightarrow \infty \).

Hence, by continuity of the functions \( f_j(t) \) over \( T^k \), it follows from inequality (31) that

\[
\lim_{k \rightarrow \infty} g_j^k(t) = \lim_{k \rightarrow \infty} h_j(t^k) = \lim_{k \rightarrow \infty} f_j(t^k) = f_j(t) \tag{39}
\]

That is, \( \lim_{k \rightarrow \infty} g_j^k(t^k) = \lim_{k \rightarrow \infty} h_j(t^k) = \lim_{k \rightarrow \infty} f_j(t^k) = f_j(t) \tag{40} \)

Consequently, from (30), (31), (32), (37), and (40), the conclusion can be drawn as follows:

\[
\lim_{k \rightarrow \infty} \Psi(t^k) = \lim_{k \rightarrow \infty} \Psi_f(t^k) = \lim_{k \rightarrow \infty} \Psi(t^k) = \Psi(t) \tag{41}
\]

As shown above, the function \( \Psi(t) \) has tight upper bound \( \Psi(t) \) and tight lower bound \( \Psi(t) \).

Let \( T^u \) be the set of accumulation points of \( \{t^k\} \) and denote \( T^* \) by the region of argmin_{t \in D} \Psi(t), where \( D \not= \emptyset \) is the feasible space of the problem \( (P2) \).

**Theorem 3.** The proposed algorithm either terminates within finite iterations at the incumbent solution being an optimal solution to the problem \( (P2) \) or an infinite sequence of the branch and bound tree is generated such that

\[
\text{LB} = \lim_{k \rightarrow \infty} \text{LB}_k
= \min_{t \in D} \Psi(t), \tag{42}
\]

\[T^u \subseteq T^* .\]

**Proof.** For each step of iterations in the proposed algorithm, \( k = 0, 1, 2, \ldots \), we have \( \text{LB}_k \leq \min_{t \in D} \Psi(t), T^k \in \text{argmin}_{t \in Q_k} \text{LB}(T), t^k = t(T^k) \in T^k \subseteq T^0 \).

Horst [44] proved that \( \text{LB}_k \) is a nondecreasing sequence bounded with \( \min_{t \in D} \Psi(t) \), which ensures the limit \( \lim_{t \in D} \text{LB}_k \leq \min_{t \in D} \Psi(t) \) exists. Since \( \{t^k\} \) is a sequence on a compact set, it has a convergent subsequence. In addition, for any \( \tilde{t} \in T^* \), there is a sequence \( \{t^r\} \) of \( \{t^k\} \) with \( \lim_{r \rightarrow \infty} t^r = \tilde{t} \). By Tuy [43] and Lemma 1, it can be drawn that the partition for subhyperrectangle sets in Step 2 is exhaustive on \( T^k \) and the choice of subhyperrectangle to be subdivided in Step 2 is bound improving. Therefore, there is a decreasing subsequence \( \{T^q\} \subseteq T^* \) where \( T^q \) is the \( T \) space of the partition \( Q_q \) with \( t^q \in T^q \), \( \text{LB}_{q_0} = \text{LB}(T^q) = \Psi(t^q) \), and \( \lim_{q \rightarrow \infty} t^q = \tilde{t} \). From Lemma 2, we have \( \lim_{q \rightarrow \infty} \text{LB}_{q_0} = \text{LB} = \Psi(\tilde{t}) \).

Next, we prove that the condition \( \tilde{t} \in D \) is satisfied. \( T^0 \) is a closed set, such that \( \tilde{t} \in T^0 \). Proved by contradiction, the above condition can be proved. Let \( \tilde{t} \in D \); there exists \( \Psi(\tilde{t}) = \delta > 0 \). Since \( \Psi \) is a continuous function, the sequence \( \{\Psi(t^q)\} \) can be converged to \( \Psi(\tilde{t}) \). According to the convergence property of \( \Psi \), \( \exists q_0 \) such that \( |\Psi(t^q) - \Psi(\tilde{t})| < \delta \) as \( q > q_0 \). Hence, for \( q > q_0 \), \( \Psi(t^q) > 0 \) implying that the problem \( (P2) \) is infeasible and violating the assumption that \( t^q = t(T^q) \). This leads to a contradiction, and therefore, \( \tilde{t} \in D \). As a result, we obtain \( \text{LB} = \Psi(\tilde{t}) = \min_{t \in D} \Psi(t), \tilde{t} \in T^* \). And the proof of Theorem 3 is completed. \( \square \)

3.4. **Computational Complexity of the Algorithm.** In this subsection, we analyze the computational complexity of the proposed algorithm by estimating the maximum number of iterations. In the process of branching operations, we denote the diameter of the \( T^k = \{l_k, u_k\} \subseteq T^0 \) by

\[
\Delta(T^k) = \max_{j=1,2,\ldots,p} \{u_j^k - l_j^k\}. \tag{43}
\]

Meanwhile, we introduce some notations as follows.

\[
\rho = \max_{j=1,2,\ldots,p} \left( \frac{(\ln u_j^0 - \ln l_j^0)}{(u_j^0 - l_j^0)} \right), \tag{44}
\]

\[
\mu = \sum_{j=1}^{p} |y_j|. \tag{45}
\]

**Lemma 3.** For given convergence tolerance \( \varepsilon \geq 0 \), if the subhyperrectangle \( T^k \subseteq T^0 \) is satisfied with \( \Delta(T^k) \leq \varepsilon / (\mu \rho) \) at the \( k_{th} \) iteration, we have

\[
\text{UB} - \text{LB}(T^k) \leq \varepsilon, \tag{46}
\]

where UB is current global upper bound of the equivalent problem and \( \text{LB}(T^k) \) indicates the linear relaxation solution of current subhyperrectangle.

**Proof.** Assuming that \( t^k \) is the optimal solution of the linear relaxation problem, we have

\[
\Psi(t^k) = \text{LB}(T^k) \leq \text{UB} = \Psi(t). \tag{47}
\]

According to (4) and (19), we can conclude that
Theorem 4. Given the convergence tolerance $\varepsilon \geq 0$, the maximum iterations of the proposed algorithm to obtain the $\varepsilon$-globally optimal solution are

$$N = \sum_{j=1}^{p} \left\lceil \frac{\mu \rho (u_j - l_j)}{\varepsilon} \right\rceil,$$

where $\mu$ and $\rho$ are given in (44) and (45), respectively.

Proof. Suppose that, at the $k_{th}$ iteration, the sub-hyperrectangle $T^k \subseteq T^0$ is satisfying with

$$\Delta(T^k) \leq \frac{\varepsilon}{\mu \rho}.$$

Meanwhile, according to the branching operation in Step 2, we have

$$\Delta(T^k) = \frac{1}{2} \sum_{j=1}^{p} (u_j^0 - l_j^0), \quad j = 1, 2, \ldots, p.$$  \hfill (52)

From (51) and (52), we obtain

$$\frac{1}{2} \sum_{j=1}^{p} (u_j^0 - l_j^0) \leq \frac{\varepsilon}{\mu \rho}, \quad j = 1, 2, \ldots, p.$$  \hfill (53)

i.e.,

$$k_j \geq \log_2 \frac{\mu \rho (u_j^0 - l_j^0)}{\varepsilon}, \quad j = 1, 2, \ldots, p.$$  \hfill (54)

With the proof of Lemma 3, we can get that $\psi(t) - \psi^R(t^k) \leq \varepsilon$. Denote

$$k_j = \left\lceil \log_2 \frac{\mu \rho (u_j^0 - l_j^0)}{\varepsilon} \right\rceil, \quad j = 1, 2, \ldots, p.$$  \hfill (55)

After at most iterations,
we have

\[
UB - LB(T) \leq \psi(t^*) - \psi^k(t^*) \leq \epsilon.
\]  

where \(t^*\) is the optimal solution of the equivalent problem (P2). Therefore, we obtain the \(\epsilon\)-globally optimal solution. The proof of Theorem 3 is completed. \(\square\)

4. Numerical Experiment

In this section, we give some numerical experiments to verify the performance of the proposed global optimization algorithm. These experiments are carried out on a workstation running Microsoft Windows 10 with Intel(R) Xeon(R) E5-2620 2.10 GHz processor and 16 GB memory. The proposed algorithm is written in Python programming language (v3.7) and the elementary simplex method is used to solve the linear relaxation programming. The corresponding numerical results are demonstrated in Tables 1, 2, and 3.

Some exact tests in recent literatures [19, 31, 36–41] are implemented to show that the proposed algorithm can globally solve the problem (MP) effectively. For these tests, the results obtained by the proposed algorithm are illustrated in Table 1. To describe the numerical results, we use the following notations for column headers in Table 1: Iter: the number of iterations; time: the execution time in seconds; optimum: the approximation optimal value; optimal solution: the approximation optimal solution for the tests; \(\epsilon\): the accuracy tolerance value.

Table 2: Computational comparisons among the algorithm of [40] and our new algorithm for problem 1.

<table>
<thead>
<tr>
<th>(p)</th>
<th>((m, n))</th>
<th>Reference [40]</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(10, 20)</td>
<td>0.1452</td>
<td>0.0681</td>
</tr>
<tr>
<td></td>
<td>(20, 40)</td>
<td>0.1514</td>
<td>0.0660</td>
</tr>
<tr>
<td></td>
<td>(40, 80)</td>
<td>0.2441</td>
<td>0.4009</td>
</tr>
<tr>
<td></td>
<td>(60, 120)</td>
<td>0.6179</td>
<td>0.5735</td>
</tr>
<tr>
<td></td>
<td>(80, 160)</td>
<td>1.9886</td>
<td>0.8842</td>
</tr>
<tr>
<td></td>
<td>(100, 200)</td>
<td>4.1027</td>
<td>1.4379</td>
</tr>
<tr>
<td></td>
<td>(10, 20)</td>
<td>0.3205</td>
<td>0.2118</td>
</tr>
<tr>
<td></td>
<td>(20, 40)</td>
<td>0.3286</td>
<td>0.4686</td>
</tr>
<tr>
<td></td>
<td>(40, 80)</td>
<td>0.7060</td>
<td>1.0471</td>
</tr>
<tr>
<td></td>
<td>(60, 120)</td>
<td>2.7911</td>
<td>1.4290</td>
</tr>
<tr>
<td></td>
<td>(80, 160)</td>
<td>6.7688</td>
<td>1.7522</td>
</tr>
<tr>
<td></td>
<td>(100, 200)</td>
<td>14.2772</td>
<td>3.0576</td>
</tr>
<tr>
<td>3</td>
<td>(10, 20)</td>
<td>0.6654</td>
<td>0.4963</td>
</tr>
<tr>
<td></td>
<td>(20, 40)</td>
<td>1.1668</td>
<td>0.8020</td>
</tr>
<tr>
<td></td>
<td>(40, 80)</td>
<td>2.2780</td>
<td>2.1902</td>
</tr>
<tr>
<td></td>
<td>(60, 120)</td>
<td>6.5797</td>
<td>3.1215</td>
</tr>
<tr>
<td></td>
<td>(80, 160)</td>
<td>14.7688</td>
<td>3.8023</td>
</tr>
<tr>
<td></td>
<td>(100, 200)</td>
<td>46.8494</td>
<td>7.5449</td>
</tr>
<tr>
<td>4</td>
<td>(10, 20)</td>
<td>1.2609</td>
<td>0.7882</td>
</tr>
<tr>
<td></td>
<td>(20, 40)</td>
<td>1.3929</td>
<td>1.5364</td>
</tr>
<tr>
<td></td>
<td>(40, 80)</td>
<td>10.4108</td>
<td>2.7617</td>
</tr>
<tr>
<td></td>
<td>(60, 120)</td>
<td>18.3711</td>
<td>4.5107</td>
</tr>
<tr>
<td></td>
<td>(80, 160)</td>
<td>81.1816</td>
<td>7.0148</td>
</tr>
<tr>
<td></td>
<td>(100, 200)</td>
<td>114.7671</td>
<td>12.2047</td>
</tr>
<tr>
<td>5</td>
<td>(10, 20)</td>
<td>1.7287</td>
<td>1.0830</td>
</tr>
<tr>
<td></td>
<td>(20, 40)</td>
<td>6.6506</td>
<td>2.0317</td>
</tr>
<tr>
<td></td>
<td>(40, 80)</td>
<td>16.3945</td>
<td>4.4962</td>
</tr>
<tr>
<td></td>
<td>(60, 120)</td>
<td>25.2491</td>
<td>8.1542</td>
</tr>
<tr>
<td></td>
<td>(80, 160)</td>
<td>98.5674</td>
<td>12.452</td>
</tr>
<tr>
<td></td>
<td>(100, 200)</td>
<td>120.0034</td>
<td>21.0908</td>
</tr>
<tr>
<td>6</td>
<td>(10, 20)</td>
<td>19.2006</td>
<td>1.6918</td>
</tr>
<tr>
<td></td>
<td>(20, 40)</td>
<td>23.8151</td>
<td>2.4351</td>
</tr>
<tr>
<td></td>
<td>(40, 80)</td>
<td>86.1229</td>
<td>7.4351</td>
</tr>
<tr>
<td></td>
<td>(60, 120)</td>
<td>155.5248</td>
<td>13.9036</td>
</tr>
<tr>
<td></td>
<td>(80, 160)</td>
<td>278.3208</td>
<td>19.4804</td>
</tr>
<tr>
<td></td>
<td>(100, 200)</td>
<td>321.4063</td>
<td>26.9879</td>
</tr>
</tbody>
</table>

\[
N = \sum_{j=1}^{p} k_j, \quad (56)
\]

\[
= \sum_{j=1}^{p} \left[ \log_2 \frac{\mu \rho(u_j^0 - f_j^0)}{\epsilon} \right],
\]

we have

\[
UB - LB(T) \leq \psi(t^*) - \psi^k(t^*) \leq \epsilon. \quad (57)
\]
Example 3. We have (see [40])

\[
\begin{align*}
\min f(x) &= (-4x_1 - 2x_4 + 3x_5 + 21)(4x_1 + 2x_2 + 3x_3 - 4x_4 + 4x_5 - 3) \\
&\quad \times (3x_1 + 4x_2 + 2x_3 - 2x_4 + 2x_5 - 7)(-2x_1 + x_2 - 2x_3 + 2x_4 + 2x_5 + 11) \\
&\quad \times (4x_1 + 4x_2 + 5x_3 + 3x_4 + x_5 \leq 25, -x_1 - 5x_2 + 2x_3 + 3x_4 + x_5 \leq 2, \\
&\quad x_1 + 2x_2 + x_3 - 2x_4 + 2x_5 \geq 6, 4x_2 + 3x_3 - 8x_4 + 11x_5 \geq 8, \\
&\quad x_1 + x_2 + x_3 + x_4 + x_5 \leq 6, x_1 \geq 1, x_2 \geq 1, x_3 \geq 1, x_4 \geq 1, x_5 \geq 1.
\end{align*}
\]

Example 1. We have (see [36–40])

\[
\begin{align*}
\min f(x) &= (0.813396x_1 + 0.67440x_2 + 0.305038x_3 + 0.129742x_4 + 0.217796) \\
&\quad \times (0.224508x_1 + 0.063458x_2 + 0.932230x_3 + 0.528736x_4 + 0.091947) \\
&\quad \times 0.488509x_1 + 0.063565x_2 + 0.945686x_3 + 0.210704x_4 \leq 3.562809, \\
&\quad -0.324014x_1 - 0.501754x_2 - 0.719204x_3 + 0.099562x_4 \leq -0.052215, \\
&\quad 0.445225x_1 - 0.346896x_2 + 0.637939x_3 - 0.257623x_4 \leq 0.427920, \\
&\quad -0.202821x_1 + 0.647361x_2 + 0.920135x_3 - 0.983091x_4 \leq 0.840950, \quad (59)
\end{align*}
\]

Example 2. We have (see [37–41])

\[
\begin{align*}
\min f(x) &= (0.201862x_1 + 0.820444x_2 - 0.305441x_3 - 0.180123x_4 \leq -1.353686, \\
&\quad -0.515399x_1 - 0.424820x_3 + 0.897498x_4 + 0.187268x_4 \leq 2.137251, \\
&\quad -0.591515x_1 + 0.060581x_2 - 0.427365x_3 + 0.579388x_4 \leq -0.290987, \\
&\quad 0.423524x_1 + 0.940496x_2 - 0.437944x_3 - 0.742941x_4 \leq 0.373620, \\
&\quad x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0.
\end{align*}
\]
Example 4. We have (see [40])
\[
\begin{align*}
\min & \quad (3x_1 - 2x_2 - 2)^{(2/3)} (x_1 + 2x_2 + 2)^{(2/5)} \\
\text{s.t.} & \quad 2x_1 - x_2 \geq 2, \\
& \quad x_1 - 2x_2 \leq 2, \\
& \quad x_1 + x_2 \leq 5, \\
& \quad 3 \leq x_1 \leq 5, \\
& \quad 1 \leq x_2 \leq 3.
\end{align*}
\]

Example 5. We have (see [19, 36, 40])
\[
\begin{align*}
\min & \quad (c_1^T x + d_1)(c_2^T x + d_2), \\
\text{s.t.} & \quad Ax = b, x \geq 0,
\end{align*}
\]
where
\[
b = (81, 72, 9, 9, 9, 8, 8)^T, \quad d_1 = 0, \quad d_2 = 0,
\]
\[
c_1 = \left(1, 0, \frac{1}{9}, 0, 0, 0, 0, 0, 0, 0\right)^T, \\
\]
\[
c_2 = \left(0, 1, \frac{1}{9}, 0, 0, 0, 0, 0, 0, 0\right)^T,
\]
\[
A = \begin{pmatrix} 9 & 9 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 1 & 8 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 8 & 8 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 7 & 1 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 1 & 7 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 1 & 1 & 7 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}
\]
\[
(63)
\]

Example 6. We have [31]
\[
\begin{align*}
\min & \quad (x_1 - 2x_2 - 2)^{(2/3)} (x_1 + 2x_2 + 2)^{(2/5)} \\
\text{s.t.} & \quad 2x_1 - x_2 \geq 2, \\
& \quad x_1 - 2x_2 \leq 2, \\
& \quad x_1 + x_2 \leq 5, \\
& \quad 3 \leq x_1 \leq 5, \\
& \quad 1 \leq x_2 \leq 3.
\end{align*}
\]

As shown in Table 1, compared with the other algorithms [19, 31, 36–41], it can be seen that the proposed algorithm can solve Examples 1–6 with fewer iterations and less time.

In order to illustrate the performance of the proposed algorithm in the complex environment, we perform a series of random problems as follows.

Problem 1. We get (see [40])
\[
\begin{align*}
\min & \quad \prod_{j=1}^p c_j^T x, \\
\text{s.t.} & \quad x \in X = \{x | Ax \leq b, x \in [0, 1]\},
\end{align*}
\]

Problem 2. We get (see [21, 45, 46])
\[
\begin{align*}
\min & \quad \prod_{j=1}^p (c_j^T x + 1), \text{s.t.} x \in X = \{x | Ax \leq b, x \geq 0\}.
\end{align*}
\]

In problems (1) and (2), each element of the vectors \(c_j\) is pseudorandom number generated in \([0, 1]\), all of the elements \(a_{ij}\) of the constraint matrix \(A\) are randomly generated in the interval \([-1, 1]\), and \(b_i, i = 1, 2, \ldots, m\) is generated by \(b_i = \sum_{j=1}^n a_{ij} + 2\eta\), where \(\eta\) is pseudorandom number generated in \([0, 1]\).

For the numerical experiments for problems (1) and (2), we perform 10 different random instances at every different scale and record the statistics information of these results in Table 2 and 3, respectively. To compare the performance of the algorithm, we set tolerance accuracy value \(\epsilon = 10^{-6}\).

In Tables 2 and 3, Avg.Time and Avg.Iter represent the average CPU times in seconds and the average number of iterations. Std.Time and Std.Iter illustrate the standard deviation of CPU times and the standard deviation of the number of iterations.

According to the computational results shown in Table 2, when \(p\) is fixed at 2, 3, 4, 5, 6, and 7 in turn, it is obvious that the elapsed CPU time of the proposed algorithm is better than what is mentioned by Zhang and Gao [40]. On the other hand, when \(p\) is smaller than 4, the number of iterations is approximately the same as that of Zhang and Gao [40]. When \(p\) is greater than 4, the number of iterations will be greatly reduced. That is, the proposed algorithm can effectively solve larger scale problems (problem 1).

Table 3 illustrates that the proposed algorithm can solve problem (2) with less computational time and fewer iterations. Meanwhile, the stability of the proposed algorithm is as good as that of Shen and Huang [46]. In addition, it is clear that the average CPU time and average iteration of the algorithm increase linearly and slowly with the scale of the problem when \(p\) is constant.

From the above numerical results, it can be concluded that the proposed algorithm can solve all examples with effectiveness and robustness.

5. Conclusion

In this paper, we study a class of multiplicative programming problems (MP) and propose a new image space branch and bound algorithm. In this algorithm, using the transformation of the logarithmic function, we transform the problem (MP) into the problem (P2). Then, based on the linear relaxation technique, we construct the linear relaxation programming problem for the problem (P2). Different from the existing algorithm, the branch operations of the proposed algorithm take place in \(p\)-dimensional image space which brings great benefits for improving the computational
efficiency of the proposed algorithm. In addition, the $\epsilon$-global convergence of the proposed algorithm is achieved at most $\sum_{j=1}^{m} \lfloor \log_2 (\mu p_j (t^j_0 - t^j_0) / \epsilon) \rfloor$ iterations. The numerical results demonstrate the feasibility and the efficiency of our algorithm. Our future work is to improve the universality of generalized linear multiplicative problems.

**Conflicts of Interest**

The authors declare that they have no conflicts of interest.

**Acknowledgments**

This work was supported by the Key Scientific and Technological Project of Henan Province (20210211035 and 202102210385), Collaborative Education Project of Industry University Cooperation of Ministry of Education (201901225003), and New Engineering Research and Practice Project of Ministry of Education of China (E-DSJ20201113).

**References**


