mathematics

MDPI

Article

A New Global Optimization Algorithm for a Class of Linear Fractional Programming

X. Liu [1], Y.L. Gao [2,3,*], B. Zhang [1] and F.P. Tian [2]

1 School of Mathematics and Statistics, Ningxia University, Yinchuan 750021, China; linhxiaoyu911@163.com (X.L.); zbsdx121@163.com (B.Z.)
2 Ningxia Province Cooperative Innovation Center of Scientific Computing and Intelligent Information Processing, North Minzu University, Yinchuan 750021, China; fptian808@163.com
3 Ningxia Province Key Laboratory of Intelligent Information and Data Processing, North Minzu University, Yinchuan 750021, China
* Correspondence: gaoyuelin@263.net; Tel.: +86-139-9510-0900

Received: 26 August 2019; Accepted: 16 September 2019; Published: 19 September 2019

Abstract: In this paper, we propose a new global optimization algorithm, which can better solve a class of linear fractional programming problems on a large scale. First, the original problem is equivalent to a nonlinear programming problem: It introduces p auxiliary variables. At the same time, p new nonlinear equality constraints are added to the original problem. By classifying the coefficient symbols of all linear functions in the objective function of the original problem, four sets are obtained, which are I_i^+, I_i^-, J_i^+ and J_i^-. Combined with the multiplication rule of real number operation, the objective function and constraint conditions of the equivalent problem are linearized into a lower bound linear relaxation programming problem. Our lower bound determination method only needs $e_i^T x + f_i \neq 0$, and there is no need to convert molecules to non-negative forms in advance for some special problems. A output-space branch and bound algorithm based on solving the linear programming problem is proposed and the convergence of the algorithm is proved. Finally, in order to illustrate the feasibility and effectiveness of the algorithm, we have done a series of numerical experiments, and show the advantages and disadvantages of our algorithm by the numerical results.

Keywords: global optimization; fractional programming; branch and bound; output-space; linear relaxation

1. Introduction

Fractional programming is an important branch of nonlinear optimization and it has attracted interest from researchers for several decades. The sum of linear ratios problem is a special class of fractional programming problem with wide applications, such as for transportation schemes, as well as finding applications in economics [1], investment and production control [2–4], and multi-objective portfolios [5]. The primary challenges in solving linear fractional programming (LFP) arise from a lack of useful properties (convexity or otherwise) and from the number of ratios and the dimension of decision space. Theoretically, it is NP-hard [6,7]. In addition, for a problem LFP, there may be several local optimal solutions [8], which interferes with finding the global optimal solution and increases the difficulty of the problem. It is therefore worthwhile to study this kind of problem. In this paper, we shall investigate the following linear fractional programming problem:

$$LFP := \min\ f(x) = \sum_{i=1}^{p} \frac{c_i^T x + d_i}{e_i^T x + f_i} \quad s.t. \quad Ax \leq b, x \geq 0.$$

where the feasible domain $X = \{x \in R^n | Ax \leq b, x \geq 0\}$ is n-dimensional, nonempty, and bounded; $p \geq 2$, $A \in R^{m \times n}$, $b \in R^m$, $c_i \in R^n$, $d_i \in R$, $e_i \in R^n$, $f_i \in R$ and $e_i^T x + f_i \neq 0$.

In the application of practical problems, p usually does not exceed 10. At present, many algorithms have been proposed to solve the *LFP* problem with a limited number of ratios. For instance, In 1962, Charnes et al. gave an effective elementary simplex method in the case of $p = 1$ [9]. On the premise of $p = 2$, Konno proposed one similar parametric elementary simplex method on the basis of reference [9], which can be used to solve large-scale problems [10]. When $p = 3$, Konno et al. constructed an effective heuristic algorithm by developing the parameter simplex algorithm [11]. When $p > 3$, Shen et al. reduced the original nonconvex programming problem to a series of linear programming problems by using equivalent transformation and linearization techniques to achieve the purpose of solving the linear fraction problem with coefficients [12]. Nguyen and Tuy considered a unified monotonic approach to generalized linear fractional programming [13]. Benson presented a simplicial branch-and-bound duality-bounds algorithm by applying the Lagrangian duality theory [6]. Jiao et al. gave a new interval reduced branch-and-bound algorithm for solving the global problem of linear ratio and denominator outcome space [14]. By exploring a well-defined nonuniform mesh, Shen et al. solved an equivalent optimization problem and proposed a complete polynomial time approximation algorithm [15]. In the same year, Hu et al. proposed a new branch-and-bound algorithm for solving the low-dimensional linear fractional programming [16]. Shen et al. introduced a practicable regional division and reduction algorithm for minimizing the sum of linear fractional functions over a polyhedron [17]. Through using a suitable transformation and linearization technique, Zhang and Wang proposed a new branch and bound algorithm with two reducing techniques to solve the generalized linear fractional programming [18]. By adopting the exponent transformation technique, Jiao et al. proposed a branch and bound algorithm of three-level linear relaxation to solve the generalized polynomial ratios problem with coefficients [19]. Based on the image space where the objective function is easy to deal with in a certain direction, Falk J E et al. transformed the problem into an "image space" by introducing new variables, and then analyzed and solved the linear fractional programming [20]. Gao Y et al. transformed the original problem into an equivalent bilinear programming problem, and used the convex envelope and concave envelope of bilinear functions to determine the lower bound of the optimal value of the original problem, and then propose a branch and bound algorithm [21]. By dividing the box where the decision variables are located, Ying Ji et al. proposed a new deterministic global optimization algorithm by relaxing the denominator on each box [22]. Furthermore, according to references [23,24], there are other algorithms that can be used to solve the *LFP* problem.

In this article, a new branch-and-bound algorithm based on the branch of output-space is proposed for globally solving the *LFP* problem. To do this, an equivalent optimization problem (*EOP*) is presented. Next, the objective function and constraint functions of the equivalence problem are relaxed using four sets (i.e., I_I^+, I_I^-, J_I^+, J_I^-) and the multiplication rules for real number operations. Based on this operation, a linear relaxation programming problem that provides a reliable lower bound for the original problem is constructed. Finally, a new branch-and-bound algorithm for the *LFP* problem is designed. Compared with the methods mentioned above (e.g., [9–15,17,18,23,24]), the goal of this research is three-fold. First of all, the lower bound of the subproblem of each node can be achieved easily, solely by solving linear programs. Secondly, the performance of the algorithm is based on the difference between the number of decision variables n and the number p of ratios. Thirdly, the problem in this article is more general than those considered in [14,17,18], since we only require $e_i^T x + f_i \neq 0$ and don't need to convert $c_i^T x + d_i < 0$ to $c_i^T x + d_i \geq 0$ for each i. However, the problem solved by our model must ensure that every decision variable is non-negative, which is also a limitation of the problem we study. In the end, the computational results of a problem with a large number of ratio terms are shown below to illustrate the feasibility and validity of the proposed algorithm.

This paper is organized as follows. In Section 2, the *LFP* problem is changed to the equivalent non-convex programming problem *EOP*. Section 3 shows how to construct a linear relaxation problem

of *LFP*. In Section 4, we give the branching rules on a hyper-rectangle. In Section 5, an output-space branch and bound algorithm is presented and its convergence is established. Section 6 introduces some existing test examples in the literature, and gives the calculation results and numerical analysis. Finally, the method of this paper is briefly reviewed, and the extension of this method to multi-objective fractional programming is prospected.

2. The Equivalence Problem of LFP

In order to establish the equivalence problem, we introduce p auxiliary variables and let $t_i = \frac{1}{e_i^T x + f_i}$, $i = 1, 2, \ldots, p$. The upper and lower bounds of t_i are referred to by $\bar{t}_i$ and $\underline{t}_i$, respectively. Then, we calculate the following linear programming problems:

$$\underline{m}_i = \min_{x \in X \cap H} e_i^T x + f_i, \quad \overline{m}_i = \max_{x \in X \cap H} e_i^T x + f_i.$$

So, we have

$$\frac{1}{\overline{m}_i} = \underline{t}_i \leq t_i = \frac{1}{e_i^T x + f_i} \leq \bar{t}_i = \frac{1}{\underline{m}_i}.$$

The hyper-rectangle of t can be denoted as follows:

$$H = [\underline{t}, \bar{t}], \underline{t} = (\underline{t}_1, \underline{t}_2, \cdots, \underline{t}_p)^T, \bar{t} = (\bar{t}_1, \bar{t}_2, \cdots, \bar{t}_p)^T.$$

Similarly, for the sub-hyper-rectangle $H^k \subseteq H$ that will be used below, the following definitions are given:

$$H^k = [\underline{t}^k, \bar{t}^k], \underline{t}^k = (\underline{t}_1^k, \underline{t}_2^k, \cdots, \underline{t}_p^k)^T, \bar{t}^k = (\bar{t}_1^k, \bar{t}_2^k, \cdots, \bar{t}_p^k)^T.$$

Finally, the *LFP* problem can be further translated into the following equivalent optimization problem:

$$EOP := \min \ f(x,t) = \sum_{i=1}^{p} (c_i^T x + d_i) t_i, \quad s.t. \begin{cases} (e_i^T x + f_i) t_i = 1, \ i = 1, 2, \cdots, p, \\ x \in X = \{x \in R^n | Ax \leq b, x \geq 0\}, \\ t \in H. \end{cases}$$

Theorem 1. *The feasible solution x^* is a global optimal solution of the LFP problem if and only if the EOP problem attaches to the global optimal solution (x^*, t^*), and for every $i = 1, 2, \ldots p$ we have equation $t_i^* = \frac{1}{e_i^T x^* + f_i}$.*

Proof of Theorem 1. If x^* is a globally optimal solution for the problem *LFP*, we have $t_i^* = \frac{1}{e_i^T x^* + f_i}$, $i = 1, 2, \ldots p$, Thus (x^*, t^*) is the feasible solution and the objective function value $f(x^*)$ of *EOP*, respectively. Let (x, t) be any feasible solution to problem *EOP*. We have

$$t_i = \frac{1}{e_i^T x + f_i}, i = 1, 2, \ldots p,$$

which means

$$f(x^*) = \sum_{i=1}^{p} \frac{c_i^T x^* + d_i}{e_i^T x^* + f_i} = \sum_{i=1}^{p} (c_i^T x^* + d_i) t_i^*.$$

Using the optimality of x^*,

$$\sum_{i=1}^{p} (c_i^T x^* + d_i) t_i^* = f(x^*, t^*) \leq f(x, t) = \sum_{i=1}^{p} (c_i^T x + d_i) t_i.$$

Hence, by $x^* \in X$ and $t_i = \frac{1}{e_i^T x^* + f_i}$, a global optimal solution (x^*, t^*) of problem *EOP* can be found.

On the other hand, problem *EOP* can also be solved and its optimal solution (x^*, t^*) obtained. Let

$$t_i^* = \frac{1}{e_i^T x^* + f_i}, i = 1, 2, \ldots p,$$

and then we have

$$f(x^*) = \sum_{i=1}^{p} \frac{c_i^T x^* + d_i}{e_i^T x^* + f_i} = \sum_{i=1}^{p} (c_i^T x^* + d_i) t_i^*.$$

Let $t_i = \frac{1}{e_i^T x + f_i}, i = 1, 2, \ldots p$, for any feasible solution x of *LFP*. Then, (x, t) is a feasible solution to problem *EOP* and the objective function is $f(x)$. According to the optimality of (x^*, t^*) and the feasibility of x, we have

$$f(x) \geq \sum_{i=1}^{p} (c_i^T x^* + d_i) t_i^* = f(x^*).$$

As $x^* \in X$, according to the above inequalities, for x^* is a global optimal solution of problem *LFP*. Thus, the *LFP* problem is equivalent to *EOP*. □

3. A New Linear Relaxation Technique

In this section, we will show how to construct a linear relaxation programming (*LRP*) for problem *LFP*. In the following, for the convenience of expression, denote:

$$I_i^+ = \{j | e_{ij} > 0, j = 1, 2, \ldots, n\}, \quad J_i^+ = \{j | c_{ij} > 0, j = 1, 2, \ldots, n\},$$
$$I_i^- = \{j | e_{ij} < 0, j = 1, 2, \ldots, n\}, \quad J_i^- = \{j | c_{ij} < 0, j = 1, 2, \ldots, n\},$$
$$\Phi_i(x, t) = (e_i^T x_j + f_i) t_i = \sum_{j=1}^{n} e_{ij} t_i x_j + f_i t_i, \ \Psi_i(x, t) = (c_i^T x_j + d_i) t_i = \sum_{j=1}^{n} c_{ij} t_i x_j + d_i t_i.$$

Then, based on the above discussion, we have

$$\begin{aligned} \Psi_i(x, t) &= t_i (\sum_{j=1}^{n} c_{ij} x_j + d_i) = \sum_{j \in I_i^+} c_{ij} t_i x_j + \sum_{j \in I_i^-} c_{ij} t_i x_j + d_i t_i, \\ &\geq \sum_{j \in I_i^+} c_{ij} \underline{t}_i x_j + \sum_{j \in I_i^-} c_{ij} \bar{t}_i x_j + d_i t_i = \underline{\Psi}_i(x, t). \end{aligned}$$

$$\begin{aligned} \Phi_i(x, t) &= t_i (\sum_{j=1}^{n} e_{ij} x_j + f_i) = \sum_{j \in I_i^+} e_{ij} t_i x_j + \sum_{j \in I_i^-} e_{ij} t_i x_j + f_i t_i, \\ &\geq \sum_{j \in I_i^+} e_{ij} \underline{t}_i x_j + \sum_{j \in I_i^-} e_{ij} \bar{t}_i x_j + f_i t_i = \underline{\Phi}_i(x, t). \end{aligned}$$

$$\begin{aligned} \Phi_i(x, t) &= t_i (\sum_{j=1}^{n} e_{ij} x_j + f_i) = \sum_{j \in I_i^+} e_{ij} t_i x_j + \sum_{j \in I_i^-} e_{ij} t_i x_j + f_i t_i, \\ &\leq \sum_{j \in I_i^+} e_{ij} \bar{t}_i x_j + \sum_{j \in I_i^-} e_{ij} \underline{t}_i x_j + f_i t_i = \overline{\Phi}_i(x, t). \end{aligned}$$

Obviously, $f(x, t) = \sum_{i=1}^{p} \Psi_i(x, t) \geq \sum_{i=1}^{p} \underline{\Psi}_i(x, t) = \underline{f}(x, t)$, $\underline{f}(x, t)$ is a lower bound function of $f(x, t)$.

Finally, we obtain a linear relaxation programming problem LRP of problem EOP by loosening the feasible region of the equivalent problem:

$$(\text{LRP}) := \min \ \underline{f}(x,t) = \sum_{i=1}^{p} \underline{\Psi}_i(x,t), \quad s.t. \begin{cases} \underline{\Phi}_i(x,t) \le 1, i = 1,2,\cdots,p, \\ \overline{\Phi}_i(x,t) \ge 1, i = 1,2,\ldots,p, \\ t \in H = \{t \in R^p | \underline{t}_i \le t_i \le \overline{t}_i, i = 1,2,\ldots p\}, \\ x \in X = \{x \in R^n | Ax \le b, x \ge 0\}. \end{cases}$$

At the same time, the linear relaxation subproblem LRP^k of problem EOP on sub-hyper-rectangle $H^k \subseteq H$ is :

$$(\text{LRP}^{\text{k}}) := \min \ \underline{f}(x,t) = \sum_{i=1}^{p} \underline{\Psi}_i(x,t), \quad s.t. \begin{cases} \underline{\Phi}_i(x,t) \le 1, i = 1,2,\cdots,p, \\ \overline{\Phi}_i(x,t) \ge 1, i = 1,2,\ldots,p, \\ t \in H^k = \{t \in R^p | \underline{t}_i^k \le t_i \le \overline{t}_i^k, i = 1,2,\ldots p\}, \\ x \in X = \{x \in R^n | Ax \le b, x \ge 0\}. \end{cases}$$

According to the above, assume that when the algorithm iterates to step k, we only need to solve problem LRP^k, whose optimal value $v(LRP^k)$ is a lower bound of the global optimum value $v(EOP^k)$of problem EOP on rectangle $H^k \subseteq H$. The optimal value $v(LRP^k)$ is also an effective lower bound of the global optimum value $v(LFP^k)$ of the original problem LFP on H^k, i.e., $v(LRP^k) \le v(EOP^k) = v(LFP^k)$.

Therefore, problem LRP^k is solved—its optimal value is obtained, which is a lower bound of the global optimum of problem LFP on rectangle H^k. The updated method of the upper bound will be explained in detail in Remark 4.

4. Branching Process

In order to facilitate the branch of the algorithm, we adopt the idea of dichotomy, and give an adaptive hyper-rectangular partition method, which depends on ω. Let $H^k = [\underline{t}^k, \overline{t}^k] \subseteq H^0 = H$ denote the current hyper-rectangle to be divided, the corresponding optimal solution of the problem LFP^k is represented by x^k and the corresponding optimal solution of the linear relaxation problem LRP^k is represented by (x^k, t^k), respectively. It is obvious that $x^k \in X$ and $t^k \in H^k$. The following forms of dissection will be performed on $H^k = [\underline{t}^k, \overline{t}^k]$:

(i): Calculate $\omega = \max\{(t_i^k - \underline{t}_i^k)(t_i^k - \overline{t}_i^k) : i = 1,2,\cdots,p\}$, if $\omega = 0$, let $\overline{t}_\mu^k - \underline{t}_\mu^k = \max\{\overline{t}_i^k - \underline{t}_i^k : i = 1,2,\cdots,p\}$, then $t_\mu^k = \frac{\underline{t}_\mu^k + \overline{t}_\mu^k}{2}$; otherwise, find the first $t_j^k \in arg\max \omega$ and let $t_\mu^k = t_j^k$.

(ii): Note that $\widehat{t} = (t_1^k, t_2^k, \cdots, t_{i-1}^k, t_\mu^k, t_{i+1}^k, \cdots, t_p^k)^T$. Using the point $\widehat{t}$, the rectangular H^k is divided into two sub-super-rectangular $H^{k1} = [\underline{t}^{k1}, \overline{t}^{k1}]$ and $H^{k2} = [\underline{t}^{k2}, \overline{t}^{k2}]$, then the sub-super-rectangles H^{k1} and H^{k2}, which are respectively:

$$H^{k1} = \prod_{i=1}^{\mu-1} [\underline{t}_i^k, \overline{t}_i^k] \times [\underline{t}_\mu^k, t_\mu^k] \times \prod_{i=\mu+1}^{p} [\underline{t}_i^k, \overline{t}_i^k],$$

$$H^{k2} = \prod_{i=1}^{\mu-1} [\underline{t}_i^k, \overline{t}_i^k] \times [t_\mu^k, \overline{t}_\mu^k] \times \prod_{i=\mu+1}^{p} [\underline{t}_i^k, \overline{t}_i^k].$$

Remark 1. *When $\omega = 0$, t^k is located at the lower left vertex or upper right vertex of the hyper-rectangle H^k, we divide the longest edge of the rectangle in a uniform dichotomous way. When $\omega \neq 0$, the dividing method depends on the position of the t^k in the hyper-rectangle H^k, and the selected μ edge is as wide as possible and ensures that t_μ^k is as close to the midpoint of the edge as possible.*

Remark 2. *The advantage of this method of dividing the hyper-rectangle is that it increases the diversity of hyper-rectangle segmentation, but to some extent, it increases the amount of extra computation. Other rules may have better performance.*

5. Output-Space Branch-and-Bound Algorithm and Its Convergence

To allow a full description of the algorithm, when the algorithm iterates to step k, we make the following representation of the associated notation: H^k is the hyper-rectangle to be thinned for the current iteration step; Q is the set of all feasible solutions to LFP; Ω is the remaining sets of hyper-rectangles after pruning; U^k is the upper bound of the global optimal value of the LFP problem when the algorithm iterates to the step k; L^k is the lower bound of the global optimal value of the LFP problem when the algorithm iterates to the step k; $L(H^k)$ represents the optimal function value of problem LRP^k on H^k and (x^k, t^k) is its corresponding optimal solution.

Using the above, a description of the output-space branch-and-bound algorithm for solving the problem LFP is as follows.

Step 1. Set the tolerance $\epsilon > 0$. Construct the initial hyper-rectangle $H^0 = H = [\underline{t}, \overline{t}]$. Solve the linear programming problem LRP^0 on super-rectangular H^0. The corresponding optimal solution and optimal value are recorded as (x^0, t^0) and $L(H^0)$, respectively. Then, $L^0 = L(H^0)$ is the initial lower bound of the global optimal value of LFP. The initial upper bound is $U^0 = f(x^0)$. If $U^0 - L^0 \leq \epsilon$, then stop, a ϵ-global optimal solution $x^* = x^0$ of problem LFP is found. Otherwise, set $\Omega = \{H^0\}$, $F = \emptyset$, the initial iteration number $k = 1$, and transfer to Step 2.

Step 2. If $U^k - L^k \leq \epsilon$, then stop the iteration of the algorithm, output the current global optimal solution x^* of the LFP problem and the globally optimal value $f(x^*)$; Otherwise, go to Step 3.

Step 3. The super-rectangle H^k, which corresponds to the current lower bound L^k, is selected, in Ω, i.e., $L^k = L(H^k)$.

Step 4. Using the rectangular branching process in Section 3, H^k is divided into two sub-rectangles: H^{k1} and H^{k2} that satisfy $H^{k1} \cap H^{k2} = \emptyset$. For all $L(H^{ki}) < U^k$, set $F = F \cup \{H^{ki}\}$, $Q = Q \cup \{x^i\}$ $(i \in \{1, 2\})$. If $F = \emptyset$, go to Step 3. Otherwise, set $\Omega = \Omega \backslash H^k \cup F$, and continue.

Step 5. Let $U^k = \min\{U^k, \min\{f(x) : x \in Q\}\}$. If $U^k = \min\{f(x) : x \in Q\}$, the current optimal solution is $x^* \in arg\min\{f(x), x \in Q\}$; Let $L^k = \min\{L(H) : H \in \Omega\}$; Set $k := k + 1$, $F = \emptyset$, $Q = \emptyset$, and go to Step 2.

Remark 3. *The branching target of our branch and bound algorithm is p-dimensional output-space, so our algorithm can be called OSBBA.*

Remark 4. *It can be seen from Step 4 and Step 5 that the number of elements in Q does not exceed two in each iterative step, and at the same time, only two function values are calculated in Step 5 to update the upper bound.*

Remark 5. *In Step 4, we save the super-rectangle H^{ki} of $L(H^{ki}) < U^k$ into Ω after each branch, which implies the pruning operation of the branching algorithm.*

Remark 6. *The convergence rate of the algorithm OSBBA. is related to the optimal accuracy and the initial hyper-rectangle H^0. It can be seen from Theorem 5 below that the convergence rate of the algorithm OSBBA is proportional to the size of the accuracy ϵ and inversely proportional to the diameter length of the initial hyper-rectangle H^0. In general, the accuracy is given in advance, and the convergence rate mainly depends on the diameter length of the initial hyper-rectangle H^0.*

Theorem 2. *Let $\varepsilon_i = \overline{t_i} - \underline{t_i}$, for each $i \in \{1, 2, \cdots, p\}$, or $\forall x \in X, t \in H$, if $\varepsilon_i \to 0$, we have $\Phi_i(x, t) - \underline{\Phi}_i(x, t) \to 0$, $\overline{\Phi}_i(x, t) - \Phi_i(x, t) \to 0$, $\Psi_i(x, t) - \underline{\Psi}_i(x, t) \to 0$, $f(x) - \underline{f}(x, t) \to 0$.*

Proof of Theorem 2. For every $\forall x \in X, t \in H$, by merging (1) and (2), we have:

$$\begin{aligned}
\Phi_i(x,t) - \underline{\Phi}_i(x,t) &= \sum_{j=1}^{n} e_{ij}t_i x_j + f_i t_i - (\sum_{j\in I_i^+} e_{ij}\underline{t}_i x_j + \sum_{j\in I_i^-} e_{ij}\bar{t}_i x_j + f_i t_i),\\
&= \sum_{j=1}^{n} e_{ij}t_i x_j - (\sum_{j\in I_i^+} e_{ij}\underline{t}_i x_j + \sum_{j\in I_i^-} e_{ij}\bar{t}_i x_j),\\
&\le \sum_{j\in I_i^+} e_{ij}|t_i - \underline{t}_i|x_j + \sum_{j\in I_i^-} |e_{ij}||\bar{t}_i - t_i|x_j,\\
&\le \sum_{j\in I_i^+} e_{ij}|\bar{t}_i - \underline{t}_i|x_j + \sum_{j\in I_i^-} |e_{ij}||\bar{t}_i - \underline{t}_i|x_j,\\
&= |\bar{t}_i - \underline{t}_i| \cdot \sum_{j=1}^{n} |e_{ij}|x_j \le N_i \cdot |\bar{t}_i - \underline{t}_i|.
\end{aligned} \tag{1}$$

$$\begin{aligned}
\overline{\Phi}_i(x,t) - \Phi_i(x,t) &= \sum_{j\in I_i^-} e_{ij}\underline{t}_i x_j + \sum_{j\in I_i^+} e_{ij}\bar{t}_i x_j + f_i t_i - (\sum_{j=1}^{n} e_{ij}t_i x_j + f_i t_i),\\
&= \sum_{j\in I_i^-} e_{ij}\underline{t}_i x_j + \sum_{j\in I_i^+} e_{ij}\bar{t}_i x_j - \sum_{j=1}^{n} e_{ij}t_i x_j,\\
&\le \sum_{j\in I_i^-} |e_{ij}||t_i - \underline{t}_i|x_j + \sum_{j\in I_i^+} e_{ij}|\bar{t}_i - t_i|x_j,\\
&\le \sum_{j\in I_i^-} |e_{ij}||\bar{t}_i - \underline{t}_i|x_j + \sum_{j\in I_i^+} e_{ij}|\bar{t}_i - \underline{t}_i|x_j,\\
&= |\bar{t}_i - \underline{t}_i| \cdot \sum_{j=1}^{n} |e_{ij}|x_j \le N_i \cdot |\bar{t}_i - \underline{t}_i|.
\end{aligned} \tag{2}$$

and

$$\begin{aligned}
\Psi_i(x,t) - \underline{\Psi}_i(x,t) &= \sum_{j=1}^{n} c_{ij}t_i x_j + d_i t_i - (\sum_{j\in J_i^+} c_{ij}\underline{t}_i x_j + \sum_{j=J_i^-} c_{ij}\bar{t}_i x_j + d_i t_i),\\
&= \sum_{j=1}^{n} c_{ij}t_i x_j - (\sum_{j\in I_i^+} c_{ij}\underline{t}_i x_j + \sum_{j\in J_i^-} c_{ij}\bar{t}_i x_j),\\
&\le \sum_{j\in J_i^+} c_{ij}|t_i - \underline{t}_i|x_j + \sum_{j\in J_i^-} |c_{ij}||\bar{t}_i - t_i|x_j,\\
&\le \sum_{j\in J_i^+} c_{ij}|\bar{t}_i - \underline{t}_i|x_j + \sum_{j\in J_i^-} |c_{ij}||\bar{t}_i - \underline{t}_i|x_j,\\
&= |\bar{t}_i - \underline{t}_i| \cdot \sum_{j=1}^{n} |c_{ij}|x_j \le N_i \cdot |\bar{t}_i - \underline{t}_i|.
\end{aligned} \tag{3}$$

where $N_i = \max\{\sum_{j=1}^{n} |e_{ij}|\overline{x}_j, \sum_{j=1}^{n} |c_{ij}|\overline{x}_j\}, i = 1, \ldots, p$. On the one hand, $x \in X$ is bounded and $0 \le x_j \le \overline{x}_j$ for each $j = 1, 2, \cdots, n$. Thus $\varepsilon_i \to 0$, $N \cdot |\bar{t}_i - \underline{t}_i| \to 0$, besides $\Phi_i(x,t) - \underline{\Phi}_i(x,t) \to 0$, $\overline{\Phi}_i(x,t) - \Phi_i(x,t) \to 0$, $\Psi_i(x,t) - \underline{\Psi}_i(x,t) \to 0$.

On the other hand,

$$f(x) - \underline{f}(x,t) = \sum_{i=1}^{p} [\Psi_i(x,t) - \underline{\Psi}_i(x,t)]. \tag{4}$$

By combining Inequalities (3) and Equation (4), then $f(x) - \underline{f}(x,t) \to 0$. Therefore, the theorem holds. □

According to Theorem 2, we can also know that the super-rectangle of t is thinning gradually and the relaxed feasible region progressively approaches the original feasible region by operation of the algorithm.

Theorem 3. *(a) If the algorithm terminates within finite iterations, a globally optimal solution for LFP is found.*

(b) If the algorithm generates an infinite sequence in the iterative process, then any accumulation point of the infinite sequence $\{x^k\}$ is a global optimal solution of the problem LFP.

Proof of Theorem 3. (a) If the algorithm is finite, assume it stops at the kth iteration, $k > 1$. From the termination rule of Step 2, we know that $U^k - L^k \leq \epsilon$, which implies that

$$f(x^k) - L^k \leq \epsilon. \tag{5}$$

Assuming that the global optimal solution is x^*, we know that

$$U^k = f(x^k) \geq f(x^*) \geq L^k. \tag{6}$$

hence, combining inequalities (5) and (6), we have

$$f(x^k) + \epsilon \geq f(x^*) + \epsilon \geq L^k + \epsilon \geq f(x^k). \tag{7}$$

and then part (a) has been proven.

(b) If the iteration of the algorithm is infinite, and in this process, an infinite sequence $\{x^k\}$ of feasible solutions for the problem LFP is generated by solving the problem LRP^k, the sequence of feasible solutions for the corresponding linear relaxation problem is $\{(x^k, t^k)\}$. According to Steps 3–5 of the algorithm, we have

$$L^k = \underline{f}(x^k, t^k) \leq f(x^*) \leq f(x^k) = U^k, k = 1, 2, \cdots. \tag{8}$$

Because the series $\{L^k = \underline{f}(x^k, t^k)\}$ is nondecreasing and bounded, and $\{U^k = f(x^k)\}$ is decreasing and bounded, they are convergent sequences. Taking the limit on both sides of (8), we have

$$\lim_{k\to\infty} \underline{f}(x^k, t^k) \leq f(x^*) \leq \lim_{k\to\infty} f(x^k). \tag{9}$$

Then, $L = \lim_{k\to\infty} \underline{f}(x^k, t^k)$, $U = \lim_{k\to\infty} f(x^k)$, and Formula (9) becomes

$$L \leq f(x^*) \leq U. \tag{10}$$

Without loss of generality, assume that the rectangular sequence $\{H^k = [\underline{t}^k, \overline{t}^k]\}$ satisfies $t^k \in H^k$ and $H^{k+1} \in H^k$. In our algorithm, the rectangles are divided continuously into two parts of equal width, then $\lim_{k\to\infty} H^k = t^*$, and in the process, a sequence $\{t^k\}$ of t will be generated, obviously, $\lim_{k\to\infty} t^k = t^*$, and also generate a sequence $\{x^k\}$ that satisfies $\lim_{k\to\infty} x^k = x^*$, because of the continuity of function $f(x)$ and Formula (10). So, the sequence $\{x^k\}$, of which any accumulation point x^* is a global optimal solution of the LFP problem. □

From Theorem 3, we know that the algorithm in this paper is convergent, and then we use Theorems 4 and 5 to show that the convergence rate of our algorithm is related to the size of p. For the detailed proof of Theorem 4, see [25], and other concepts in the theorem are derived from [25]. In addition, we encourage readers to understand [25] in detail.

As the sub-hyper-rectangles obtained by our branch method are not necessarily congruent, take $\delta(H) = \max\{\delta(H^l) : l \in \{1, 2, \cdots, s\}\}$, where the definition of s is given below. The definition of $\delta(H^l)$ is the same as that of Notation 1 in [25], which represents the diameter of hyper-rectangle H^l. Therefore, $\delta(H)$ represents the maximum diameter of the s hyper-rectangles. In order to connect well with the content of this paper, we adjust the relevant symbols and reinterpret.

Theorem 4. *Consider the big cube small cube algorithm with a bounding operation which has a rate of convergence of $q \geq 1$. Furthermore, assume a feasible super-rectangle H and the constants $\epsilon, C > 0$ as before. Moreover, we assume the branching process which splits the selected super-rectangle along each side, i.e., into $s = 2^r$ smaller super-rectangles. Then the worst case number of iterations for the big cube small cube method can be bounded from above by*

$$\sum_{v=0}^{z} 2^{r \cdot v} \quad where \quad z = \lceil log_2 \frac{\delta(H)}{(\epsilon/C)^{1/q}} \rceil, \quad \delta(H) = \max\{\delta(H^l) : l \in \{1, 2, \cdots, s\}\}. \tag{11}$$

Proof of Theorem 4. The proof method is similar to the Theorem 2 in [25] and is thus omitted. □

In Theorem 4, r represents the spatial dimension of the hyper-rectangle to be divided. At the same time, Tables 1 and 2 in [25] show that $q = 1$ is the worst case, that is, the most times the algorithm needs to subdivide hyper-rectangles during iteration. For the convenience of the discussion, we assume $q = 1$, and give Theorem 5 to show that the convergence rate of our algorithm is related to the size of p.

Theorem 5. *For the algorithm OSBBA, it is assumed that for a feasible hyper-rectangle H_p, there is a fixed positive number of C_p and the accuracy ϵ. In addition, we also assume that the branching process will eventually divide the hyper-rectangle into $s = 2^p$ small hyper-rectangles. Then, in the worst case, the number of iterations of the OSBBA algorithm when dividing the hyper-rectangle H_p can be expressed by the following formula:*

$$\sum_{v=0}^{z_p} 2^{p \cdot v} \quad where \quad z_p = \lceil log_2 \frac{C_p \cdot \delta(H_p)}{\epsilon} \rceil, \quad \delta(H_p) = \max\{\delta(H_p^l) : l \in \{1, 2, \cdots, s\}\}. \tag{12}$$

We call the convergence rate of the algorithm OSBBA, $O(p)$.

Proof of Theorem 5. We order "$r = p$", "$C = C_p$", "$q = 1$", "$z = z_p$" and "$H = H_p$" in Theorem 4, the proof method is similar to Theorem 4, and the reader can refer to [25]. □

In addition, for the algorithms in [18,26–29], they subdivide the n-dimensional hyper-rectangle H_n. Similar to Theorem 4, when they divide the hyper-rectangle H_n, the number of iterations in the worst case can also be expressed by the following formula:

$$\sum_{v=0}^{z_n} 2^{n \cdot v} \quad where \quad z_n = \lceil log_2 \frac{\delta(H_n)}{(\epsilon/C_n)^{1/q_n}} \rceil, \quad \delta(H_n) = \max\{\delta(H_n^l) : l \in \{1, 2, \cdots, s\}\}. \tag{13}$$

where "n", "C_n", "q_n", "z_n" and "H_n" correspond to "r", "C", "q", "z" and "H" in (11). We also record the convergence rate of the algorithm in [18,26–29] as $O(n)$.

By means of Equations (12) and (13), when $p \ll n$, the following conclusions are drawn:

(i): If $z_p \leq z_n$, then, $\sum_{v=0}^{z_p} 2^{p \cdot v} \leq \sum_{v=0}^{z_n} 2^{p \cdot v} \ll \sum_{v=0}^{z_n} 2^{n \cdot v}$.

(ii): If $z_p > z_n$, there must be a positive number $N \geq \lfloor \frac{z_p}{z_n} p \rfloor + 1$ so that $p < \frac{z_p}{z_n} p < N$ holds, which means that when $N \ll n$ implies that $p < \frac{z_p}{z_n} p < N \ll n$, $pz_p \ll nz_n$ also holds, then:

$$\begin{aligned}\sum_{v=0}^{z_n} 2^{n\cdot v} - \sum_{v=0}^{z_p} 2^{p\cdot v} &= \frac{2^{n(z_n+1)}-1}{2^n-1} - \frac{2^{p(z_p+1)}-1}{2^p-1},\\ &= \frac{(2^{n(z_n+1)}-1)(2^p-1) - (2^{p(z_p+1)}-1)(2^n-1)}{(2^n-1)(2^p-1)},\\ &= \frac{2(2^{n+p-1}-1)(2^{nz_n}-2^{pz_p}) + 2^n - 2^p}{(2^n-1)(2^p-1)} \gg 0.\end{aligned}$$

Both conclusions (i) and (ii) can show that when $p \ll n$, the following formula is established: $\sum_{v=0}^{z_p} 2^{p\cdot v} \ll \sum_{v=0}^{z_n} 2^{n\cdot v}$.

Remark 7. *In Formula (12) of Theorem 5, $q = 1$, while the q_n in Formula (13) does not specify the size, which means that $O(p)$ is compared with $O(n)$ in the case of slowest convergence, but in the case of $p \ll n$, there will always be (i) and (ii), which is a clearer indication of $O(p) \ll O(n)$.*

It can be seen that the size of $O(p)$ and $O(n)$ is exponential growth, but the size of p in $O(p)$ is generally not more than 10, and $p \ll n$, so our algorithm *OSBBA* has an advantage in solving large-scale problems in the case of $p \leq 10$ and $p \ll n$. The experimental analysis of several large-scale random examples below will also be referred to again.

6. Numerical Examples

Now, we give several examples and a random calculation example to prove the validity of the branch-and-bound algorithm in this paper.

We coded the algorithms in Matlab 2017a and ran the tests in a computer with an Intel(R) Core(TM)i7-4790s processor of 3.20 GHz, 4 GB of RAM memory, under the Microsoft Windows 7 operational system. In solving the *LRPs*, we use the simplex method in the linprog command in Matlab 2017a.

In Tables 1–9, the symbols of the table header are respectively: x^*, the optimal solution of the *LFP* problem; $f(x^*)$, the optimal value of the objective function; Iter, the number of iterations on problems 1–11; Ave.Iter, the average number of iterations on problems 12–13; ϵ, tolerance; Time: the CPU running time of problems 1–11; Ave.Time, the average CPU running time of problems 12–13; p, the number of linear fractions in the objective function; m, the number of linear constraints; n, the dimension of the decision variable; SR, the success rate of the algorithm in calculating problem 12 . When the number of Ave.Time or Ave.Iter shows "–", it means that the algorithm fails to calculate when solving the problem.

Example 1 ([15,18]).

$$\min \quad \frac{-x_1+2x_2+2}{3x_1-4x_2+5} + \frac{4x_1-3x_2+4}{-2x_1+x_2+3} \quad s.t. \begin{cases} x_1+x_2 \leq 1.5, \\ x_1-x_2 \leq 0, \\ 0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1. \end{cases}$$

Example 2 ([18]).

$$\min \quad 0.9 \times \frac{-x_1+2x_2+2}{3x_1-4x_2+5} + (-0.1) \times \frac{4x_1-3x_2+4}{-2x_1+x_2+3} \quad s.t. \begin{cases} x_1+x_2 \leq 1.5, \\ x_1-x_2 \leq 0, \\ 0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1. \end{cases}$$

Example 3 ([17,21]).

$$\min \ \frac{3x_1+5x_2+3x_3+50}{3x_1+4x_2+5x_3+50}+\frac{3x_1+4x_2+50}{4x_1+3x_2+2x_3+50}+\frac{4x_1+2x_2+4x_3+50}{5x_1+4x_2+3x_3+50}$$
$$s.t.\begin{cases} 2x_1+x_2+5x_3\le 10,\\ x_1+6x_2+2x_3\le 10,\\ 9x_1+7x_2+3x_3\ge 10,\\ x_1,x_2,x_3\ge 0. \end{cases}$$

Example 4 ([17,22]).

$$\begin{aligned}\min \ & -\Big(\frac{4x_1+3x_2+3x_3+50}{3x_2+3x_3+50}+\frac{3x_1+4x_3+50}{4x_1+4x_2+5x_3+50}\\ & +\frac{x_1+2x_2+5x_3+50}{x_1+5x_2+5x_3+50}+\frac{x_1+2x_2+4x_3+50}{5x_2+4x_3+50}\Big)\end{aligned}$$
$$s.t.\begin{cases} 2x_1+x_2+5x_3\le 10,\\ x_1+6x_2+3x_3\le 10,\\ 5x_1+9x_2+2x_3\le 10,\\ 9x_1+7x_2+3x_3\le 10,\\ x_1\ge 0,x_2\ge 0,x_3\ge 0. \end{cases}$$

Example 5 ([17,21,24]).

$$\begin{aligned}\min \ & \frac{4x_1+3x_2+3x_3+50}{3x_2+3x_3+50}+\frac{3x_1+4x_3+50}{4x_1+4x_2+5x_3+50}\\ & +\frac{x_1+2x_2+4x_3+50}{x_1+5x_2+5x_3+50}+\frac{x_1+2x_2+4x_3+50}{5x_2+4x_3+50}\end{aligned}$$
$$s.t.\begin{cases} 2x_1+x_2+5x_3\le 10,\\ x_1+6x_2+2x_3\le 10,\\ 9x_1+7x_2+3x_3\ge 10,\\ x_1,x_2,x_3\ge 0. \end{cases}$$

Example 6 ([17,22]).

$$\min \ -\Big(\frac{3x_1+5x_2+3x_3+50}{3x_1+4x_2+5x_3+50}+\frac{3x_1+4x_2+50}{4x_1+3x_2+2x_3+50}+\frac{4x_1+2x_2+4x_3+50}{5x_1+4x_2+3x_3+50}\Big)$$
$$s.t.\begin{cases} 6x_1+3x_2+3x_3\le 10,\\ 10x_1+3x_2+8x_3\le 10,\\ x_1,x_2,x_3\ge 0. \end{cases}$$

Example 7 ([17,21]).

$$\min \ \frac{37x_1+73x_2+13}{13x_1+13x_2+13}+\frac{63x_1-18x_2+39}{13x_1+26x_2+13} \quad s.t.\begin{cases} 5x_1-3x_2=3,\\ 1.5\le x_1\le 3. \end{cases}$$

Example 8 ([12,14]).

$$\max \quad \frac{4x_1+3x_2+3x_3+50}{3x_2+3x_3+50}+\frac{3x_1+4x_2+50}{4x_1+4x_2+5x_3+50} + \frac{x_1+2x_2+5x_3+50}{x_1+5x_2+5x_3+50}+\frac{x_1+2x_2+4x_3+50}{5x_2+4x_3+50}$$
$$s.t. \begin{cases} 2x_1+x_2+5x_3 \le 10, \\ x_1+6x_2+3x_3 \le 10, \\ 5x_1+9x_2+2x_3 \le 10, \\ 9x_1+7x_2+3x_3 \le 10, \\ x_1 \ge 0, x_2 \ge 0, x_3 \ge 0. \end{cases}$$

Example 9 ([12,14,23]).

$$\max \quad \frac{37x_1+73x_2+13}{13x_1+13x_2+13}+\frac{63x_1-18x_2+39}{-13x_1-26x_2-13} + \frac{13x_1+13x_2+13}{63x_1-18x_2+39}+\frac{13x_1+26x_2+13}{-37x_1-73x_2-13}$$
$$s.t. \begin{cases} 5x_1-3x_2=3, \\ 1.5 \le x_1 \le 3. \end{cases}$$

Example 10 ([14,23,24]).

$$\max \quad \frac{4x_1+3x_2+3x_3+50}{3x_2+3x_3+50}+\frac{3x_1+4x_3+50}{4x_1+4x_2+5x_3+50} + \frac{x_1+2x_2+5x_3+50}{x_1+5x_2+5x_3+50}+\frac{x_1+2x_2+4x_3+50}{5x_2+4x_3+50}$$
$$s.t. \begin{cases} 2x_1+x_2+5x_3 \le 10, \\ x_1+6x_2+2x_3 \le 10, \\ 9x_1+7x_2+3x_3 \ge 10, \\ x_1 \ge 0, x_2 \ge 0, x_3 \ge 0. \end{cases}$$

Example 11 ([18]).

$$\max \quad \frac{3x_1+4x_2+50}{3x_1+5x_2+4x_3+50}-\frac{3x_1+5x_2+3x_3+50}{5x_1+5x_2+4x_3+50} - \frac{x_1+2x_2+4x_3+50}{5x_2+4x_3+50}-\frac{4x_1+3x_2+3x_3+50}{3x_2+3x_3+50}$$
$$s.t. \begin{cases} 6x_1+3x_2+3x_3 \le 10, \\ 10x_1+3x_2+8x_3 \le 10, \\ x_1 \ge 0, x_2 \ge 0, x_3 \ge 0. \end{cases}$$

As can be seen from Table 1, our algorithm can accurately obtain the global optimal solution of these 11 low-dimensional examples, which shows the effectiveness and feasibility of this algorithm. However, compared with other algorithms in the literature, the effect of this algorithm is relatively poor. This is because the method of constructing the lower bound is simple and easy to operate, and the branching operation is performed on the p-dimensional output-space. At the same time, the algorithm of this paper has no super-rectangular reduction technology, which makes the approximation of solving the low dimensional problem worse. We also note that the number p of ratios in these 11 examples is not larger than the dimension n of the decision variable, and our algorithm requires that p is much smaller than n, which is why our algorithm is not effective in solving these examples. With the continuous updating and progress of computers, the gap between our algorithm and other methods in solving these 11 low-dimensional examples can be bridged, and the needs of society mainly focus on the high-dimensional problems under $p \ll n$. Therefore, we only use Examples 1–11 to illustrate

the effectiveness and feasibility of our algorithm, and the numerical results can also show that the algorithm is convergent. When our algorithm is applied to higher dimensional problems, the effect gradually improves, as can be seen from the numerical results of Examples 12 and 13 in Tables 2–9.

Table 1. Comparison of results in Examples 1–11.

Example	Methods	x^*	$f(x^*)$	*Iter*	*Time*	ϵ
1	[15]	(0.00,0.2817)	1.6232	43	4.3217	10^{-8}
1	[18]	(0.00,0.2839)	1.6232	65	0.9524	10^{-8}
1	*OSBBA*	(0.00,0.2839)	1.6232	1983	40.3528	10^{-8}
2	[18]	(0.0,1.0)	3.5750	1	0.0561	10^{-9}
2	*OSBBA*	(0.0,1.0)	3.5750	12	0.1626	10^{-9}
3	[17]	(5.0000,0.0000,0.0000)	2.8619	16	0.1250	10^{-3}
3	[21]	(5.0000,0.0000,0.0000)	2.8619	12	28.2943	10^{-4}
3	*OSBBA*	(5.0000,0.0000,0.0000)	2.8619	379	8.4163	10^{-4}
4	[17]	(1.1111,0.0000,0.0000)	−4.0907	185	3.2510	10^{-2}
4	[22]	(1.0715,0,0)	−4.0874	17	-	10^{-6}
4	*OSBBA*	(1.11111111,0,0)	−4.0907	70	1.8196	10^{-6}
5	[17]	(0.0000,1.66666667,0.0000)	3.7109	8	0.1830	10^{-3}
5	[21]	(0.0000,1.6667,0.0000)	3.7087	5	4.1903	10^{-4}
5	[24]	(0.0000,0.625,1.875)	4.0000	58	2.9686	10^{-4}
5	*OSBBA*	(0.0000,1.66666667,0.0000)	3.7109	169	4.2429	10^{-6}
6	[17]	(0,0.333333,0)	−3.0029	17	0.1290	10^{-3}
6	[22]	(0,0.33329,0)	−3.0000	30		10^{-6}
6	*OSBBA*	(0,0.333333,0)	−3.0029	2090	50.8226	10^{-6}
7	[17]	(1.5000,1.5000)	4.9125	56	1.0870	10^{-3}
7	[21]	(1.5000,1.5000)	4.9125	113	201.6260	10^{-4}
7	*OSBBA*	(1.5000,1.5000)	4.9125	460	8.7944	10^{-4}
8	[12]	$(1.1111, 0, -3.33067 \times 10^{-5})$	4.0907	3	0.0000	10^{-6}
8	[14]	(1.11111, 0.00000, 0.00000)	4.0907	2	0.0020	10^{-6}
8	*OSBBA*	(1.11111, 0.00000,0.00000)	4.0907	42	1.1433	10^{-6}
9	[12]	(3.0, 4.0)	3.2916	9	0.0000	10^{-6}
9	[14]	(3.0, 4.0)	3.2916	2	0.0017	10^{-6}
9	[23]	(3.0, 4.0)	3.2916	78	0.0000	10^{-6}
9	*OSBBA*	(3.0,4.0)	3.2916	693	16.5359	10^{-6}
10	[14]	(5.0,0.0,0.0)	4.4285	2	0.0018	10^{-6}
10	[23]	(5.0,0.0,0.0)	4.4285	35	0.0000	10^{-4}
10	[24]	(0.0, 0.625, 1875)	4.0000	58	2.9686	10^{-4}
10	*OSBBA*	(5.0,0.0,0.0)	4.4285	61	1.6153	10^{-6}
11	[18]	(0.0,3.3333,0.0)	1.9000	8	0.1389	10^{-6}
11	*OSBBA*	(0.0,3.3333,0.0)	1.9000	402	6.8145	10^{-6}

Example 12.

$$\min \sum_{i=1}^{p} \frac{\sum_{j=1}^{n} c_{ij}x_j + d_i}{\sum_{j=1}^{n} e_{ij}x_j + f_i} \quad s.t. \begin{cases} \sum_{j=1}^{n} a_{qj}x_j \le b_q, \ q = 1, 2, \cdots, m, \\ x_j \ge 0, \ j = 1, 2, \cdots, n, \end{cases}$$

where p is a positive integer , c_{ij}, e_{ij}, a_{qj} are randomly selected on the interval [0,1], and set $b_q = 1$ for all q. All constant terms of denominators and numerators are the same number, which randomly generated in [1,100]. This agrees that the random number is generated in [18]. First of all, when the dimension is not more than 500, we generate 10 random examples for each group (p, m, n), and use the algorithm OSBBA and the algorithm in [18] to calculate the same example respectively, and then record the average number of iterations and the average CPU running time of these 10 examples in Table 2, respectively. Secondly, when the dimension n is not less than 1000, it is noted that the algorithm in [18] needs to solve 2n linear programming problems when determining the initial hyper-rectangle, and the search space of each linear programming is at least thousands of dimensions, which is very time-consuming. Note that when (p, m, n) = (10, 200, 500), the CPU time is close to 1200 s. Therefore,

in the case where the dimension n is not less than 1000, We generate only five random examples and specify that the algorithm is considered to fail when the calculated time exceeds 1200 s. On the premise of recording the average number of iterations and the average CPU running time, the success rate of the five high-dimensional examples is also added, which is presented in Table 3.

Table 2. The results of random calculations for Example 12. Ave.Iter, the average number of iterations on problems 12–13; Ave.Time, the average CPU running time of problems 12–13; *SR*, the success rate of the algorithm in calculating problem 12.

p	*m*	*n*	*OSBBA*			*Reference* [18]		
			Ave.Iter	*Ave.Time*	*SR*	*Ave.Iter*	*Ave.Time*	*SR*
5	30	30	3.3	0.1636	100%	2.5	0.0866	100%
5	50	50	4.2	0.2090	100%	2.7	0.1818	100%
5	100	100	4.9	0.3075	100%	3.3	1.7679	100%
10	30	30	7.7	0.4194	100%	2.8	0.1440	100%
10	50	50	8.2	0.6288	100%	4.7	0.3852	100%
10	100	100	12.8	0.8931	100%	6.3	2.7120	100%
5	50	100	5.1	0.2191	100%	2.8	0.4551	100%
5	100	300	18.4	2.3497	100%	72.5	49.1801	100%
5	200	500	16.7	11.1795	100%	19.9	328.1308	100%
10	50	100	480.7	11.4817	100%	50.6	17.0826	100%
10	100	300	818.5	95.7513	100%	1064.4	836.2154	100%
10	200	500	435.8	281.8541	100%	1784.5	1126.2541	100%

First of all, by comparing the lower bound subproblem in [18] with the lower bound subproblem in our algorithm, we can know that the lower bound subproblem of algorithm *OSBBA* only makes use of the information of the upper and lower bounds of the denominator of p ratios in the process of construction, while in the process of constructing the lower bound, the information of the upper and lower bounds of the decision variables is also used, which requires the calculation of $2n$ linear programming problems in the initial iterative step. Compared with the method in [18], the algorithm *OSBBA* only needs to calculate the $2p$ linear programming problems in the initialization stage, and does not need to calculate the upper and lower bounds of any fractal denominator. It can be seen that when p is much less than n, [18] will spend a lot of time in calculating $2n$ linear programming problems. The number of branches is often particularly large when the number of iterations is greater than 1, so that a large number of child nodes will be produced on the branch and bound tree, which not only occupies a large amount of computer memory space but also takes a lot of time. However, the performance of our algorithm *OBSBA* is the opposite. In real life, the size of p is usually not greater than 10, therefore, the number of subproblems that will need to be solved is usually small in the process of branching iteration. Compared with the method in [18], the amount of computer memory occupied is not very large.

Secondly, from the results of Table 2, we can see that the computational performance of [18] in solving small-scale problems is better than that of our algorithm *OSBBA*. However, it can be clearly seen that when the dimension of the problem is higher than 100, its computational performance gradually weakens. It can also be seen that when the dimension of the problem is above 100, the computational power of the method in [18] is inferior to our algorithm *OSBBA*. The computational performance of the algorithm *OSBBA* is closely related to the size of p, and the smaller the p is, the shorter the computing time is. For the algorithm in [18], its computational performance has a very important relationship with the size of n. The larger the n, the more time the computer consumes. It is undeniable that the method in [18] has some advantages in solving small-scale problems. However, in solving large-scale problems, Table 3 shows that the algorithm *OSBBA* is always superior to the algorithm in [18]. Especially when the dimension is more than 500, the success rate of our algorithm to find the global optimal solution in 1200 s is higher than that in [18]. This is the advantage of algorithm *OSBBA*.

Table 3. The results of random calculations for Example 12.

p	m	n	*OSBBA*			*Reference* [18]		
			Ave.Iter	*Ave.Time*	*SR*	*Ave.Iter*	*Ave.Time*	*SR*
5	100	1000	7.5	5.7765	80%	–	–	0%
5	200	2000	14.0	96.2968	80%	–	–	0%
5	300	3000	18.5	555.5875	80%	–	–	0%
10	100	1000	19.2	127.5658	60%	–	–	0%
10	200	2000	32.1	649.6260	60%	–	–	0%
10	300	3000	39.4	964.1816	60%	–	–	0%
5	300	5000	6.1	594.2337	60%	–	–	0%
5	400	8000	2.3	785.0338	60%	–	–	0%
5	500	10,000	1.4	913.8774	40%	–	–	0%
10	300	5000	2.1	216.3840	40%	–	–	0%
10	400	8000	2.4	1095.3918	40%	–	–	0%
10	500	10,000	2.3	1180.6531	20%	–	–	0%

In addition, for Example 12, we also use the same test method as the [18] to compare the algorithm *OSBBA* with the internal solver *BMIBNB* of MATLAB toolbox *YALMIP* [26], where we only record the the average CPU running time and the success rate of the two algorithms and display them in Tables 4 and 5.

Table 4. The results of random calculations for Example 12.

p	m	n	*OSBBA*		*BMIBNB*	
			Ave.Time	*SR*	*Ave.Time*	*SR*
5	30	30	0.1911	100%	0.0840	100%
5	50	50	0.2106	100%	0.1117	100%
5	100	100	0.9178	100%	0.7412	100%
10	30	30	4.7010	100%	2.2465	100%
10	50	50	5.8519	100%	3.9418	100%
10	100	100	33.9510	100%	11.5741	100%
5	50	100	0.5447	100%	6.9531	100%
5	100	300	4.1197	100%	78.3507	100%
5	200	500	69.9101	100%	403.4631	100%
10	50	100	108.1924	100%	10.4275	100%
10	100	300	115.5843	100%	149.4162	100%
10	200	500	238.9797	100%	806.4061	100%

Table 5. The results of random calculations for Example 12.

p	m	n	*OSBBA*		*BMIBNB*	
			Ave.Time	*SR*	*Ave.Time*	*SR*
5	100	1000	38.6369	100%	–	0%
5	200	2000	161.0308	80%	–	0%
5	300	3000	250.7251	80%	–	0%
10	100	1000	167.5179	80%	–	0%
10	200	2000	369.3976	60%	–	0%
10	300	3000	603.8334	20%	–	0%
5	300	5000	484.2731	60%	–	0%
5	400	8000	697.9532	60%	–	0%
5	500	10,000	907.1948	40%	–	0%
10	300	5000	716.1021	40%	–	0%
10	400	8000	1105.1437	40%	–	0%
10	500	10,000	1177.9841	20%	–	0%

As can be seen from Tables 4 and 5, the $BMIBNB$ is more effective than the $OSBBA$ when solving the small-scale problem, but it is sensitive to the size n of the decision variable, especially when n exceeds 100, the CPU time of the computer is suddenly increased. The algorithm $OSBBA$ is less affected by n, but for small-scale problems, the computational performance of the algorithm $OSBBA$ is very sensitive to the number p of the linear fractions. For large-scale problems, Table 5 shows similar results as Table 3.

Example 13.

$$\min \sum_{i=1}^{p} \frac{\sum_{j=1}^{n} c_{ij}x_j + d_i}{\sum_{j=1}^{n} e_{ij}x_j + f_i} \quad s.t. \begin{cases} \sum_{j=1}^{n} a_{qj}x_j \leq b_q, \ q = 1, 2, \cdots, m, \\ 0 \leq x_j \leq \overline{x}_j, \ j = 1, 2, \cdots, n. \end{cases}$$

In order to further illustrate the advantages of the algorithm $OSBBA$ in this paper, a large number of numerical experiments were carried out for Example 13, comparing the algorithm $OSBBA$ with the call to the commercial software package $BARON$ [27]. Through the understanding of $BARON$, we know that the operation of its branches is also carried out in the n-dimensional space. Similar to [18], we can predict that $BARON$ is quite time-consuming as the dimension increases. To simplify the comparison operation, the one of the constants of the numerator and denominator is set to 100 (i.e., $d_i = f_i = 100$) in order to successfully run $BARON$. Next, we give an upper bound $\overline{x}_j$ for each decision variable, and randomly select from the interval $[0, 10]$ together with c_{ij}, e_{ij}, a_{qj} and b_q to form a random Example 13.

We set the tolerance of the algorithm $OSBBA$ and $BARON$ to 10^3 for the sake of fairness (This is because the accuracy of the internal setting of the package $BARON$ is 10^3 and we are unable to adjust it). For each group (m, n, p), we randomly generate 10 examples, calculate the same example with the algorithm $OSBBA$ and the commercial software package $BARON$ respectively, and then record the average number of iterations and the average CPU running time of the 10 examples in Tables 6–9.

Table 6. The results of random calculations for Example 13.

m	*n*	*p*	*OSBBA*		*BARON*	
			Ave.Iter	*Ave.Time*	*Ave.Iter*	*Ave.Time*
5	10	2	36	1.1612	2	0.7473
5	20	2	79	1.9147	4	0.6075
5	30	2	114.5	2.7861	9	0.6482
5	50	2	75	1.9192	5	0.6511
5	70	2	286	5.9812	14	0.7949
5	90	2	67.5	1.6454	13	0.8312
5	100	2	156.5	3.0462	6	0.7616
5	10	3	203.5	5.1221	7	0.5021
5	20	3	624.5	14.6705	9	0.6231
5	30	3	246	6.9570	58	0.9305
5	50	3	466	11.2603	16	0.9116
5	70	3	97	4.0403	4	0.7513
5	90	3	815.4	15.5686	638.8	3.5839
5	100	3	753	13.8953	26.2	0.9445
5	10	4	896	18.8903	197	0.7966
5	10	5	138.5	3.4184	1	0.5202
5	10	6	245.5	5.6823	3	0.4332
5	20	4	472.5	9.0087	4	0.4629
5	20	5	269.5	5.8828	4	0.6038
5	20	6	4482	96.0794	90	0.8938
5	30	4	140.5	3.2252	3	0.4684
5	30	5	205	4.7040	3	0.4701
5	30	6	1404.5	31.7456	18	0.6425
10	30	5	2401	50.5017	32	0.6387
10	50	5	444	9.6959	3	0.5758

Table 6. *Cont.*

m	*n*	*p*	*OSBBA*		*BARON*	
			Ave.Iter	*Ave.Time*	*Ave.Iter*	*Ave.Time*
10	70	5	777	16.9510	36	0.9000
10	90	5	2187.5	49.8752	595	5.9809
10	100	2	64.4	1.2717	7.2	0.7844
10	100	3	223.1	4.4559	14.2	0.9466
10	100	4	594.2	12.1786	9651.2	53.4707
10	200	4	2009.2	44.0950	4363	49.0320
10	100	5	4414	88.7048	30,905.4	205.4919
10	200	5	3899	91.1614	35,481	502.3583
10	300	5	6591	176.2488	22,469	505.1200
20	300	5	1668.2	45.0485	14,140.1	307.2198
20	500	5	5091.8	165.9364	4878.2	264.4288
20	500	6	14,975.1	577.1660	6854.1	416.4063
20	500	7	62,930.1	3012.5028	7133.7	487.2844
20	500	8	3570	179.9689	36	62.7567
20	500	9	75,222	3909.7296	6740.5	637.1516
20	500	10	30,274	1504.8824	32	74.1234

As we can see from Table 6, when n is less than 100, the CPU run time (*Ave.Time*) and the iteration number (*Ave.Iter*) of our algorithm are not as good as $BARON$. In the case of $p = 2, 3$, n = 100, the CPU average running time (*Ave.Time*) and the average iteration number (*Ave.Iter*) of $BARON$ are less than our algorithm $OSBBA$. In the state of $(m, n) = (10, 100)$ and $p = 4, 5$, the algorithm $OSBBA$ is better than $BARON$. In the case of $n \geq 200$, if $p \leq 5$, the average CPU running time (*Ave.Time*) of algorithm $OSBBA$ is less than $BARON$, while at $p > 5$, the average running time of the algorithm is opposite to that of the former.

Table 7. The results of random calculations for example 13.

m	*n*	*p*	*OSBBA*		*BARON*	
			Ave.Iter	*Ave.Time*	*Ave.Iter*	*Ave.Time*
50	300	2	20	1.1057	5	4.8298
50	300	3	27	2.9405	5	5.6102
50	300	4	105	5.2798	25	28.8512
50	500	2	16	1.8027	3	12.2832
50	500	3	51	2.6929	17	23.8195
50	500	4	68	8.9815	4381	356.2157
50	700	2	37	3.8667	11	67.7368
50	700	3	58	4.9608	7738	1002.5904
50	700	4	41	12.7235	3	51.0155
50	900	2	36	5.3660	7	108.5627
50	900	3	34	7.27312	3	142.4374
50	900	4	40	14.9773	3	178.2181
50	1000	2	46	10.6561	1	131.5696
50	1000	3	82	22.04153	3	162.2937
50	1000	4	636	105.3631	19	315.2514
50	2000	2	50	36.0024	1	779.5139
50	2000	3	169	67.2509	6	1007.5297
50	2000	4	436	113.1053	29	1008.2303
50	3000	2	77	82.6399	3	304.8984
50	3000	3	123	96.3323	–	–
50	4000	2	42	85.5768	–	–
50	4000	3	156	129.1936	–	–
50	5000	2	81	93.9734	–	–
50	2000	5	370	215.7799	–	–
50	2000	6	1274	283.4462	–	–
50	2000	7	2867	366.5079	–	–
50	2000	8	3032	414.5711	–	–

Table 8. The results of random calculations for Example 13.

m	*n*	*p*	*OSBBA*		*BARON*	
			Ave.Iter	*Ave.Time*	*Ave.Iter*	*Ave.Time*
100	300	3	19.3	4.1248	2	6.7865
100	300	5	105	6.6735	3.8	11.3750
100	300	8	676.4	41.5368	6.8	14.4418
100	500	3	21.9	5.9263	3.4	20.4406
100	500	5	162.3	16.4179	6.8	35.2466
100	500	8	2812.1	235.0307	13.4	69.6685
150	500	3	15.9	7.3081	2.4	21.4519
150	500	5	58.1	8.3569	3	35.9739
150	500	8	694.4	80.6513	7.2	51.4611
200	700	3	9.3	2.7830	2	47.8042
200	700	5	45.2	10.8633	3.2	84.6023
200	700	8	118.7	40.3079	4.2	122.5232
300	700	3	4.6	1.8944	1.6	60.0400
300	700	5	10.5	5.38701	1.8	78.3503
300	700	8	91.3	28.5988	4	119.6601
300	900	3	2.1	1.8728	1.2	93.1878
300	900	5	24.5	10.8376	3.2	158.5425
300	900	8	549	175.6600	3.8	231.1125
300	1000	3	9.1	6.8319	2.2	131.6551
300	1000	5	19.1	15.6941	2.2	144.4356
300	1000	8	98.3	85.0639	3.2	220.8366

Table 9. The results of random calculation for example 13.

m	*n*	*p*	*OSBBA*		*BARON*	
			Ave.Iter	*Ave.Time*	*Ave.Iter*	*Ave.Time*
500	1000	3	1.5	3.4675	1	99.9415
500	1000	5	7.2	9.7535	2.2	147.3751
500	1000	8	24.7	30.0108	2.8	187.4586
500	2000	3	3	16.0995	1.7	496.0448
500	2000	5	6.9	23.9408	2.6	741.5934
500	2000	8	22.3	93.7039	2.7	861.7061
500	3000	3	3.5	26.4755	3.0	920.6775
500	3000	5	4.5	56.1823	–	–
500	3000	8	30	125.0527	–	–
500	4000	3	2.4	54.7006	–	–
500	4000	5	9	136.5505	–	–
500	4000	8	10.8	196.5768	–	–
500	5000	3	2	71.7622	–	–
500	5000	5	5	126.6783	–	–
500	5000	8	17.3	330.9163	–	–
500	6000	3	1.8	72.6898	–	–
500	7000	3	2.6	238.4842	–	–
500	9000	3	2.3	209.8106	–	–
500	10,000	3	3.7	319.7487	–	–

According to Tables 7–9, we can also conclude that if $p < 5$, the algorithm *OSBBA* takes significantly less than *BARON*. In Tables 8 and 9, in the case of $p \leq 8$, if $n = 300, 500$, the calculation time of algorithm *OSBBA* is significantly more than *BARON*, if $n = 700, 900, 1000, 2000, 3000$, the calculation of *BARON* takes more time than the algorithm *OSBBA*. At the same time, some "–" can be seen in Tables 7 and 9, *BARON* fails in these 10 computations, which indicates that the success rate of *BARON* in solving high-dimensional problems is close to zero, but our algorithm can still obtain the global optimal solution of the problem within a finite step, and the overall time is no more than 420 s.

In general, when $p \ll n$, our algorithm showed a good computing performance. In practical application, the size of p generally does not exceed 10. The calculation results in Table 6 show that our algorithm is not as effective as the $BARON$ algorithm in solving small problems, but it can be seen from Tables 6–9 that this algorithm has obvious advantages in solving large-scale and high-dimensional problems. At the same time, compared with $BARON$, this algorithm can also solve high-dimensional problems.

The method of the nonlinear programming in commercial software package $BARON$ comes from [29]. It is a branch and bound reduction method based on the n-dimensional space in which the decision variable x is located, which we can see from the two examples in Section 6 of [29]. In Section 5 of [29], many feasible domain reduction techniques, including polyhedral cutting techniques, are also proposed. Although $BARON$ connects many feasible domain reduction techniques when using this method, from the experimental results in this paper, it can be seen that $BARON$ is more effective than our $OSBBA$ method in solving small-scale linear fractional programming problems. $BARON$ branches on a maximum of 2^n nodes, which is exponential, while our algorithm, potentially branches on a max of 2^p nodes, a smaller-sized problem. Even if these feasible domain reductions are still valid in combination with $BARON$, when a computer runs a reduced program in these feasible domains, it also increases time consumption and space storage to a large extent. For special nonlinear programming-linear fractional programming problems, the proposed global optimization algorithm is to branch hyper-rectangles in p-dimensional space, because p is much less than n, and p is not more than 10, which ensures that the algorithm proposed by us is suitable for solving large-scale problems. As the variables that the algorithm branches are located in different spatial dimensions, $BARON$ branches on the n-dimensional decision space where the decision variable is located, and the branching process of the algorithm $OSBBA$ is carried out on the p-dimensional output-space. It can also be seen from Tables 6–9 that when the number p of ratio items is much smaller than the dimension n of the decision variable, the algorithm $OSBBA$ calculates better than $BARON$. This is because in the case of higher dimensional problems, in the process of initialization, $BARON$ needs to solve more and higher dimensional subproblems, while the algorithm $OSBBA$ only needs to solve $2p$ n-dimensional subproblems, which greatly reduces the amount of computation. This is why our algorithm $OSBBA$ branches in p-dimensional space.

In summary, in terms of the demand of real problems, the number p of ratios does not exceed 10 in the linear fractional programming problems required to be solved. At the same time, the size of p is much smaller than the dimension n of the decision variable. In the process of branching, the number of vertices of the rectangle to be divided is 2^p and 2^n, respectively. In the case of $p \ll n$, the branch of our algorithm $OSBBA$ can always be completed quickly, but the methods in the software package $BARON$ and in [18] will have a lot of branching complexity. Therefore the computation required by the branch search in R^p is more economical than that in R^n. In the case of $p \ll n$, our method is more effective in solving large-scale problems than in [18] and the software package $BARON$. At the same time, it is also noted that when the results of $OSBBA$ and $BMIBNB$ are compared, the latter is sensitive to the size of n, which once again illustrates the characteristics of the algorithm in this paper.

7. Conclusions

In this paper, a deterministic method is proposed for linear fractional programming problems. It is based on the linear relaxation problem of the positive and negative coefficient of the constructor, and the corresponding branch-and-bound algorithm $OSBBA$ is given. In Section 6, the feasibility and effectiveness of the algorithm for solving linear fractional programming problems are fully illustrated by numerical experiments, and it is also shown that our algorithm $OSBBA$ is more effective than the method in $BARON$, $BMIBNB$, and [18] when applied to high-dimensional problems in the case of $p \ll n$. In recent years, the development of multi-objective programming is becoming increasingly rapid. We can solve the problem of multi-objective linear fractional programming by combining the

ideas and methods of this paper with other approaches. In future academic research, we will also start to consider this aspect of the problem.

Author Contributions: X.L. and Y.G. conceived of and designed the study. B.Z. and X.L. performed the experiments. X.L. wrote the paper. Y.G. and F.T. reviewed and edited the manuscript. All authors read and approved the manuscript.

Funding: This research is supported by the National Natural Science Foundation of China under Grant (11961001), the Construction Project of first-class subjects in Ningxia higher Education(NXYLXK2017B09) and the major proprietary funded project of North Minzu University (ZDZX201901).

Acknowledgments: The authors are grateful to the responsible editor and the anonymous references for their valuable comments and suggestions, which have greatly improved the earlier version of this paper.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

LFP	linear fractional programming
EOP	equivalent nonlinear programming
LRP	linear relaxation programming
T	transpose of a vector or matrix

References

1. Schaible, S. Fractional programming. In *Handbook of global optimization*; Horst, R., Pardalos, P.M., eds.; Kluwer: Dordrecht, The Netherlands, 1995; pp. 495–608.
2. Falk, J.E.; Palocsay, S.W. Optimizing the Sum of Linear Fractional Functions. In *Recent Advances in Global Optimization*; Princeton University Press: Princeton, NJ, USA, 1992; pp. 221–258.
3. Konno, H.; Inori, M. Bond portfolio optimization by bilinear fractional programming. *J. Oper. Res. Soc. Jpn.* **1989**, *32*, 143–158. [CrossRef]
4. Colantoni, C.S.; Manes, R.P.; Whinston, A. Programming, profit rates and pricing decisions. *Account. Rev.* **1969**, *44*, 467–481.
5. Sawik, B. Downside risk approach for multi-objective portfolio optimization. In *Operations Research Proceedings 2011*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 191–196.
6. Benson, H.P. A simplicial branch and bound duality-bounds algorithm for the linear sum-of-ratios problem. *Eur. J. Oper. Res.* **2007**, *182*, 597–611. [CrossRef]
7. Freund, R.W.; Jarre, F. Solving the Sum-of-Ratios Problem by an Interior-Point Method. *J. Glob. Optim.* **2001**, *19*, 83–102. [CrossRef]
8. Matsui, T. NP-hardness of linear multiplicative programming and related problems. *J. Glob. Optim.* **1996**, *9*, 113–119. [CrossRef]
9. Charnes, A.; Cooper, W.W. Programming with linear fractional functionals. *Naval Res. Logist. Q.* **1962**, *9*, 181–186. [CrossRef]
10. Konno, H.; Yajima, Y.; Matsui, T. Parametric simplex algorithms for solving a special class of nonconvex minimization problems. *J. Glob. Optim.* **1991** , *1*, 65–81. [CrossRef]
11. Konno, H.; Abe, N. Minimization of the sum of three linear fractional functions. *J. Glob. Optim.* **1999**, *15*, 419–432. [CrossRef]
12. Shen, P.P.; Wang, C.F. Global optimization for sum of linear ratios problem with coefficients. *Appl. Math. Comput.* **2006**, *176*, 219–229. [CrossRef]
13. Nguyen, T.H.P.; Tuy, H. A Unified Monotonic Approach to Generalized Linear Fractional Programming. *J. Glob. Optim.* **2003**, *26*, 229–259.
14. Jiao, H.; Liu, S.; Yin, J.; Zhao, Y. Outcome space range reduction method for global optimization of sum of affine ratios problem. *Open Math.* **2016**, *14*, 736–746. [CrossRef]
15. Shen, P.P.; Zhang, T.L.; Wang, C.F. Solving a class of generalized fractional programming problems using the feasibility of linear programs. *J. Inequal.* **2017**, *2017*, 147. [CrossRef] [PubMed]

16. Hu, Y.; Chen, G.; Meng, F. Efficient Global Optimization Algorithm for Linear-fractional-programming with Lower Dimension. *Sci. Mosaic* **2017**, *1*, 11–16.
17. Shen, P.P.; Lu, T. Regional division and reduction algorithm for minimizing the sum of linear fractional functions. *J. Inequal.* **2018**, *2018*, 63. [CrossRef] [PubMed]
18. Zhang, Y.H.; Wang, C.F. A New Branch and Reduce Approach for Solving Generalized Linear Fractional Programming. *Eng. Lett.* **2017**, *25*, 262–267.
19. Jiao, H.W.; Wang, Z.K.; Chen, Y.Q. Global optimization algorithm for sum of generalized polynomial ratiosproblem. *Appl. Math. Model.* **2013**, *37*, 187–197. [CrossRef]
20. Falk, J.E.; Palocsay, S.W. Image space analysis of generalized fractional programs. *J. Glob. Optim.* **1994**, *4*, 63–88. [CrossRef]
21. Gao, Y.; Jin, S. A global optimization algorithm for sum of linear ratios problem. *Math. Appl.* **2013**, *2013*, 785–790. [CrossRef]
22. Ji, Y.; Zhang, K.C.; Qu, S.J. A deterministic global optimization algorithm. *Appl. Math. Comput.* **2007**, *185*, 382–387. [CrossRef]
23. Shi, Y. Global Optimization for Sum of Ratios Problems. Master's Thesis, Henan Normal University, Xinxiang, China, 2011.
24. Wang, C.F.; Shen, P.P. A global optimization algorithm for linear fractional programming. *Appl. Math. Comput.* **2008**, *204*, 281–287. [CrossRef]
25. SchBel, A.; Scholz, D. The theoretical and empirical rate of convergence for geometric branch-and-bound methods. *J. Glob. Optim.* **2010**, *48*, 473–495.
26. Lofberg, J. YALMIP: A toolbox for modeling and optimization in MATLAB. *IEEE Int. Symp. Comput. Aided Control Syst. Des.* **2004**, *3*, 282–289.
27. Sahinidis, N. BARON User Manual v.17.8.9[EB/OL]. Available online: http://minlp.com (accessed on 18 June 2019).
28. Sahinidis, N.V. BARON 14.3.1: Global Optimization of Mixed-Integer Nonlinear Programs. User's Manual. 2014. Available online: http://archimedes.cheme.cmu.edu/?q=baron (accessed on 13 September 2019).
29. Tawarmalani, M.; Sahinidis, N.V. A polyhedral branch-and cut approach to global optimization. *Math. Programm.* **2005**, *103*, 225–249. [CrossRef]