On Multiple-Ratio Hyperbolic 0-1 Programming Problems*

Oleg A. Prokopyev [‡] Cláudio N. Meneses [†‡]

Carlos A. S. Oliveira [§] Panos M. Pardalos [‡]

Dedicated to 65th birthday of Professor Hiroshi Konno.

Abstract. Multiple-ratio hyperbolic (fractional) 0–1 programming problems are considered. We investigate complexity issues of these problems including local search, approximability and global verification. Some aspects of linear mixed 0–1 reformulations are also discussed. In addition, we present a GRASP-based (*Greedy Randomized Adaptive Search*) heuristic for solving cardinality constrained problems.

Key words: fractional 0–1 programming, hyperbolic 0–1 programming, multiple-ratio, NP-hard, local search

1. Introduction

One of the classes of 0–1 optimization problems is the maximization of the sum of ratios of linear 0–1 functions:

$$\max_{\mathbf{x}\in\mathbb{B}^n} f(\mathbf{x}) = \sum_{j=1}^{m} \frac{a_{j0} + \sum_{i=1}^{n} a_{ji}x_i}{b_{j0} + \sum_{i=1}^{n} b_{ji}x_i}, \tag{1}$$

where $\mathbb{B}^n = \{0,1\}^n$. This problem is referred to as *hyperbolic (fractional) 0-1 programming problem*, or *multiple-ratio fractional (hyperbolic) 0–1 programming problem* [2, 8]. Usually it is assumed that for all j and $x \in \mathbb{B}^n$ the denominators in (1) are positive, i.e. $b_{j0} + \sum_{i=1}^{n} b_{ji}x_i > 0$.

*This research work was partially supported by NSF and AirForce grants. The authors would like to thank the anonymous referees for their valuable suggestions that have improved the quality of the paper.

†Supported by the Brazilian Federal Agency for Post-Graduate Education (CAPES) - Grant No. 1797-99-9.

‡Dept. of Industrial and Systems Engineering, University of Florida, 303 Weil Hall, Gainesville, FL, 32611, USA. Emails: {oap4ripe,claudio,pardalos}@ufl.edu

§School of Industrial Engineering and Management, Oklahoma State University, 322 Engineering North, Stillwater, OK 74078, USA. Email: coliv@okstate.edu

A special class of problem (1) is the so-called *single-ratio hyperbolic 0-1 programming problem*:

$$\max_{\mathbf{x}\in\mathbb{B}^n} f(\mathbf{x}) = \frac{a_0 + \sum_{i=1}^n a_i x_i}{b_0 + \sum_{i=1}^n b_i x_i}. \tag{2}$$

Applications of *constrained and unconstrained* versions of the problems (1) and (2) arise in numerous areas including but not limited to scheduling [22], query optimization in data bases and information retrieval [8], and p-choice facility location [26].

Problem (2) can be generalized if instead of linear 0–1 functions we consider multi-linear polynomials:

$$\max_{\mathbf{x}\in\mathbb{B}^n} f(\mathbf{x}) = \frac{\sum_{S\in A} a_S \prod_{i\in S} x_i}{\sum_{T\in B} b_T \prod_{j\in T} x_j}, \tag{3}$$

where $A,\ B$ are families of subsets of $\{1, 2, \ldots, n\}$.

It is easy to observe that after simple manipulations we can always reduce problem (1) to (3) and the degrees of polynomials in (3) are upper bounded by the number of ratios in (1). Note also that introducing a new binary variable for each product $\prod_{i\in S} x_i$ and $\prod_{j\in T} x_j$, problem (3) can be reformulated as an equivalent constrained single-ratio hyperbolic 0-1 programming problem. Therefore, any multiple-ratio hyperbolic 0–1 programming problem (1) can be reduced to a constrained single-ratio problem (2).

Problem (3) has very interesting applications in graph theory [19]. Consider an undirected graph $G = (V, E)$. The *density* $d(G)$ of the graph G is defined as the maximum ratio of the number of edges e_H to the number of nodes n_H over all subgraphs $H \subseteq G$, i.e.

$$d(G) = \max_{H\subseteq G} \frac{e_H}{n_H}, \tag{4}$$

where e_H and n_H are the number of edges and nodes in the subgraph H. Obviously, the problem of finding $d(G)$ can be formulated as the following hyperbolic 0–1 programming problem:

$$d(G) = \frac{1}{2} \max_{\mathbf{x}\in\mathbb{B}^{n_G},\ \mathbf{x}\neq 0} (\sum_{i=1}^{n_G}\sum_{j=1}^{n_G} a_{ij} x_i x_j) / \sum_{j=1}^{n_G} x_j, \tag{5}$$

where a_{ij} are the elements of the adjacency matrix of G and n_G is the number of nodes in G. Similar formulation can also be given for the *arboricity* $\Gamma(G)$ defined as the minimum number of edge-disjont forests into which G can be decomposed. More detailed description of these problems along with polynomial time algorithms for specific classes of (3) can be found in [19].

Algorithms for solving constrained and unconstrained versions of problems (1)-(3) include linearization techniques [15, 26, 28, 27], branch and bound methods [22, 26], network-flow [19] and approximation [9] approaches. Optimization of sums-of-ratios problems over convex sets is considered in [6, 13, 14, 21]. Extensive reviews on fractional programming can be found in [23, 24, 25].

The remainder of this paper is organized as follows. In Section 2 we consider the computational complexity of hyperbolic 0–1 programming problems. To the authors' knowledge no results on complexity of multiple-ratio hyperbolic 0–1 programming problems have been reported until recent work [20]. In Section 3 the cardinality constrained problem is discussed. Section 4 is devoted to linearization techniques. In Section 5 we present a GRASP-based heuristic for solving the cardinality constrained multiple-ratio hyperbolic 0–1 programming problems. Finally, Section 6 concludes the discussion.

2. Complexity issues

Constrained versions of problems (1) and (2), where we solve the problem subject to some linear 0–1 constraints, are clearly NP-hard since 0–1 programming is a special class of constrained hyperbolic 0–1 programming if $b_{ji} = 0$ and $b_{j0} = 1$ for $j = 1, \ldots, m$ and $i = 1, \ldots, n$.

Next we discuss complexity issues of unconstrained single- and multiple-ratio hyperbolic 0–1 programming problems (1) and (2).

It is well-known that there exists a polynomial time algorithm for solving an unconstrained single-ratio hyperbolic 0–1 programming problem (2), see Ref. [8], if the following condition holds:

$$b_0 + \sum_{i=1}^{n} b_i x_i > 0 \text{ for all } \mathbf{x} \in \mathbb{B}^n. \tag{6}$$

Note that if the term $b_0 + \sum_{i=1}^{n} b_i x_i$ can take the value zero, then problem (2) may not have a finite optimum. In the case where

$$b_0 + \sum_{i=1}^{n} b_i x_i \neq 0, \text{ for all } \mathbf{x} \in \mathbb{B}^n \tag{7}$$

holds, but the term $b_0 + \sum_{i=1}^{n} b_i x_i$ can take both negative and positive values, single-ratio problem (2) is known to be NP-hard [8]. Moreover, finding an approximate solution within any positive multiple of the (negative) optimal value is NP-hard [8]. It is also easy to observe that checking condition (7) is NP-hard since **SUBSET SUM** can be reduced to it. Multiple-ratio problem (1) remains NP-hard if $a_{j0} + \sum_{i=1}^{n} a_{ji} x_i \geq 0$, $b_{j0} + \sum_{i=1}^{n} b_{ji} x_i > 0$ for all $\mathbf{x} \in \mathbb{B}^n$ and for all $j = 1, \ldots, m$ [20].

For multiple-ratio problem conditions (6) and (7) correspond to

$$b_{j0} + \sum_{i=1}^{n} b_{ji} x_i > 0 \text{ for all } \mathbf{x} \in \mathbb{B}^n \text{ and } j = 1, \ldots, m, \tag{8}$$

and

$$b_{j0} + \sum_{i=1}^{n} b_{ji} x_i \neq 0, \text{ for all } \mathbf{x} \in \mathbb{B}^n \text{ and } j = 1, \ldots, m. \tag{9}$$

Some other aspects of complexity of unconstrained single- and multiple-ratio hyperbolic 0–1 programming problems (1) and (2) are addressed in [20]. It is shown that

(a) checking uniqueness of the solution for both problems (1) and (2) is NP-hard (we assume that only condition (9) holds);

(b) finding the global solution for single-ratio problem (2) is NP-hard even if it is known that the respective global solution is unique;

(c) multiple-ratio problem (1) remains NP-hard even if it is known that the respective global solution is unique;

(d) multiple-ratio problem (1) is PLS-complete, where PLS stands for the class of polynomial time local search problems [12];

(e) multiple-ratio problem (1) is not ϵ-approximable in polynomial time for some constant $\epsilon > 0$.

Last three results remain valid if condition (8) holds.

2.1. Improved inapproximability results for multiple-ratio problem.

For combinatorial optimization problems, where f is the corresponding objective function, an *ϵ-approximate minimal solution*, or *ϵ-minimizer*, $\epsilon \geq 1$ is usually defined as an $\mathbf{x} \in \{0,1\}^n$ such that

$$f(\mathbf{x}) \leq \epsilon \cdot Optimum.$$

The proof of the next result is based on the **SET COVER** problem known to be NP-hard [7]. The input is *a ground set* $E = \{e_1, e_2, \ldots, e_n\}$ of elements with subsets $S = \{S_1, S_2, \ldots, S_m\}$, where $S_i \subset E$ for each $i = 1, \ldots, m$. The goal is to choose the smallest collection $\overline{S} \subseteq S$ of sets whose union is E. We let $m = |S|$ and $n = |E|$.

Theorem 1 *[4] If there is some $\epsilon > 0$ such that a polynomial time algorithm can approximate set cover within $(1-\epsilon)\ln n$, then $NP \subset TIME(n^{O(\log\log n)})$. This result holds even if we restrict ourselves to set cover instances with $m \leq n$.*

In other words, **Theorem 1** state that $(1 - o(1))\ln n$ is a threshold below which set cover cannot be approximated efficiently, unless NP can be solved by a slightly superpolynomial time algorithms (for more details, please, see [4, 16]).

For problem (1) we assume that condition (8) is satisfied. Using the aforementioned result by Feige the following theorem can be proved:

Theorem 2 *If there is some $\epsilon > 0$ such that a polynomial time algorithm can approximate minimization of a multiple-ratio hyperbolic 0–1 function within $(1-\epsilon)\ln m$, where m is the number of binary variables in the objective function, then $NP \subset TIME(m^{O(\log\log m)})$.*

Proof. We reduce **SET COVER** problem to a minimization of a multiple-ratio hyperbolic 0–1 function.

We are given a ground set $E = \{e_1, \ldots, e_n\}$, and collection $S = \{S_1, \ldots, S_m\}$ of subsets of E, where $m = |S|$, $n = |E|$ and $m \leq n$. With each subset S_i we associate a binary variable x_i. With each element $e_k \in E$ we associate the following 0–1 function:

$$g_k(\mathbf{x}) = 1 - \sum_{i:e_k \in S_i} \frac{x_i}{1 + \sum_{j \neq i,\ e_k \in S_j} x_j}.$$

If the set S_i is selected then we have $x_i = 1$, otherwise $x_i = 0$. Note that for any $\overline{S} \subseteq S$ of subsets of E, if the element $e_k \in \cup\overline{S}$ then the corresponding function $g_k(\mathbf{x}) = 0$, otherwise $g_k(\mathbf{x}) = 1$.

With an instance of **SET COVER** problem we associate the following unconstrained multiple-ratio hyperbolic 0–1 programming problem

$$\min_{\mathbf{x} \in \mathbb{B}^m} f(\mathbf{x}) = \sum_{i=1}^{m} x_i + M \sum_{i=1}^{n} g_i(\mathbf{x}), \tag{10}$$

where M is a constant number such that $M > m \ln m$. It is easy to see that for any $\mathbf{x} \in \mathbb{B}^m$ $f(\mathbf{x}) \geq 0$ and if the set $\overline{S} \subseteq S$ associated with $\mathbf{x}$ *covers* E then $f(\mathbf{x}) \leq m$, otherwise $f(\mathbf{x}) > m \ln m + 1$ (by the selection of M).

Suppose next that there exists a polynomial time algorithm that can approximate (10) within $(1 - \epsilon) \ln m$. Let $\mathbf{x}^* = (x_1^*, \ldots, x_m^*)$ be an approximate solution obtained by this algorithm and S^* be a collection of sets from S associated with $\mathbf{x}^*$. Since by our assumption $\mathbf{x}^*$ is an approximate solution within $(1 - \epsilon) \ln m$ we have that

$$f(\mathbf{x}^*) \leq (1 - \epsilon) \ln m \cdot \mathit{Optimum} \leq (1 - \epsilon) \ln m \cdot m < m \ln m,$$

and, therefore, S^* covers E. It means that we obtain an approximate solution to **SET COVER**, which can not be guaranteed unless $NP \subset TIME(m^{O(\log \log m)})$. □

2.2 Complexity of problems with the fixed number of ratios.

The results proved above are valid for problems, where we do not fix the number of ratios. Another interesting question will be to consider the complexity of problem (1) with the fixed number of ratios in the objective function. Why is it important? First of all, in many real-life problems, which can be solved using formulation (1), the number of ratios m and the number of variables n correspond to different parameters of the initial problem. For example, in p-choice facility location problem (see, [26]) n is the number of possible facility locations and m is the number of customer locations. Therefore, if we know the complexity of problem (1) with the fixed number of ratios we may estimate the complexity of our initial problem, where the parameter of the problem, which

correspond to the number of ratios in (1) is fixed or small. Another interesting issue arising here is the following. We know that if condition (8) is satisfied then for $m = 1$ we have a classical case which can be solved in polynomial time. In other words, the sign of the denominator is "the borderline between between polynomial and NP-hard classes" of single-ratio problem (2) [8]. As we will see in the theorem stated below the number of ratios ($m = 1$, or $m \geq 2$) will be the borderline between between polynomial and NP-hard classes for problem (1), where condition (8) is satisfied.

Theorem 3 *If the number of ratios m in (1) is a fixed number and condition (8) is satisfied, then for $m \geq 2$ problem (1) remains NP-hard.*

Proof. In order to prove the needed result it is enough to show that problem (1) subject to condition (8) remains NP-hard for $m = 2$. We use the classical **SUBSET SUM** problem: Given a set of positive integers $S = \{s_1, s_2, \ldots, s_n\}$ and a positive integer K, does there exist a vector $\mathbf{x} \in \mathbb{B}^n$, such that

$$\sum_{i=1}^{n} s_i x_i = K? \tag{11}$$

This problem is known to be NP-complete [7].

Let M be a large constant such that $M > \sum_{i=1}^{n} s_i + K$. With the instance of the **SUBSET SUM** problem we associate the following hyperbolic 0–1 programming problem:

$$\max_{\mathbf{x}\in\mathbb{B}^n} f(\mathbf{x}) = -\frac{1}{M - (\sum_{i=1}^{n} s_i x_i - K)} - \frac{1}{M + (\sum_{i=1}^{n} s_i x_i - K)}. \tag{12}$$

Condition (8) is satisfied by the selection of M. After simple manipulations (12) can be rewritten as

$$\max_{\mathbf{x}\in\mathbb{B}^n} f(\mathbf{x}) = -\frac{2M}{M^2 - (\sum_{i=1}^{n} s_i x_i - K)^2}. \tag{13}$$

It is easy to verify that the maximum of (13) is $-\frac{2}{M}$ if and only if (11) has a solution.

□

If we replace $\sum_{i=1}^{n} s_i x_i - K$ by $\sum_{i=1}^{n} s_i x_i + K x_{n+1} - K$ in (13) and consider the following problem

$$\max_{\mathbf{x}\in\mathbb{B}^{n+1}} f(\mathbf{x}) = -\frac{2M}{M^2 - (\sum_{i=1}^{n} s_i x_i + K x_{n+1} - K)^2}, \tag{14}$$

then the following theorem can be established.

Theorem 4 *If condition (8) holds then the problem of checking if (1) has a unique solution is NP-hard. This result remains valid if the number of ratios m in (1) is a fixed number such that $m \geq 2$.*

Proof. It is easy to see that $\mathbf{x} = (0, \ldots, 0, 1)$, where $x_i = 0$ for $i = 1, \ldots, n$ and $x_{n+1} = 1$ is a solution of problem (14). Therefore, the **SUBSET SUM** problem (11) is reduced to checking if (14) has a unique solution or not. □

In [20] it was shown that if all coefficients in the objective function are integers then the multiple-ratio problem (1) with m ratios can be reduced in polynomial time to the problem with $m + 1$ ratios and unique global solution. Therefore, we can state the following result:

Theorem 5 *If the number of ratios m in (1) is a fixed number and condition (8) is satisfied, then for $m \geq 3$ problem (1) is NP-hard even if it is known that the respective global solution is unique.*

2.3 Complexity of local search.

For any point $x \in \mathbb{B}^n$ its *adjacent points* (or *neighbors*) can be defined as

$$\mathbf{x}^k = (x_1, \ldots, x_{k-1}, 1 - x_k, x_{k+1}, \ldots, x_n), \ k = 1, \ldots, n.$$

A point $\mathbf{x} \in \mathbb{B}^n$ is locally optimal if it does not have a neighbor whose function value is strictly better than $f(\mathbf{x})$. For the maximization problem it means that a point $\mathbf{x} \in \mathbb{B}^n$ is a *discrete local maximizer (dlm)* if $f(\mathbf{x}) \geq f(\mathbf{x}^k)$ for all $k = 1, \ldots, n$.

In [20] it was proved that unconstrained multiple-ratio hyperbolic 0–1 programming problem is PLS-complete. This result was shown to remain valid if $a_{j0} + \sum_{i=1}^{n} a_{ji} x_i \geq 0$, $b_{j0} + \sum_{i=1}^{n} b_{ji} x_i > 0$ for all $\mathbf{x} \in \mathbb{B}^n$ and for all $j = 1, \ldots, m$. Next we discuss complexity of finding a dlm for problem (1).

Consider again the **SUBSET SUM** problem with the following input $S = \{s_1, \ldots, s_n\}$ and K. Given the instances of S and K, we say that the subset $\widetilde{S} = \{s_{k1}, \ldots, s_{km}\} \subseteq S$ is a local minimum if and only if

$$|\sum_{s_i \in \widetilde{S}} s_i - K| \leq |\sum_{s_i \in \widetilde{S}} s_i - K + s'|$$

for all $s' \in S - \widetilde{S}$, and

$$|\sum_{s_i \in \widetilde{S}} s_i - K| \leq |\sum_{s_i \in \widetilde{S}} s_i - K - s''|$$

for all $s'' \in \widetilde{S}$. In other words, $\widetilde{S}$ is the closest to the solution among its neighborhood sets.

The following lemma was proved in [18].

Lemma 1 (Pardalos-Jha) *Given a set of integers $S = \{s_1, \ldots, s_n\}$ and an integer K, the problem of finding a local minima $\widetilde{S} = \{s_{k1}, \ldots, s_{km}\} \subseteq S$ such that $s_n, \ s_{n-1} \notin \widetilde{S}$ is NP-hard.*

This lemma allows us to consider complexity of finding dlm for problem (1) with two coordinates fixed.

Theorem 6 *Given an instance of unconstrained multiple-ratio hyperbolic 0–1 programming problem (1), the problem of finding a dlm $x^* = (x_1^*, \ldots, x_n^*)$ such that $x_{n-1}^* = x_n^* = 0$, is NP-hard. This result remains valid if condition (8) holds, and/or the number of ratios m in (1) is a fixed number such that $m \geq 2$.*

Proof.

Let $S = \{s_1, \ldots, s_n\}$ and K be an instance of the **SUBSET SUM** problem. Consider the hyperbolic 0–1 programming problem defined in (13). If x^* is a dlm of (13) with $x_{n-1}^* = x_n^* = 0$, then the subset $\widetilde{S} = \{s_j : \ x_j^* = 1\}$ is a local minimum for the **SUBSET SUM** problem.

□

Similar results for quadratic 0–1 programming problems were proved in [18].

2.4 Complexity of global verification.

For an optimization problem P, where we maximize some function $f : \Omega \to \mathbb{R}$, the *global verification decision problem* is defined as: Given an instance of P and a feasible solution $w \in \Omega$, does there exist a feasible solution $w' \in \Omega$ such that $f(w') > f(w)$?

The global verification problem is NP-complete for **MAX-SAT**, **MAX-k-SAT** ($k \geq 2$), **Vertex Cover** [1], the **Travelling Salesman Problem** [17]. For more information on global verification and related class of PGS problems (polynomial time global search) we can refer to [11].

Theorem 7 *Given an instance of unconstrained multiple-ratio hyperbolic 0–1 programming problem (1) the related global verification decision problem is NP-hard. This result remains valid if condition (8) holds, and/or the number of ratios m in (1) is a fixed number such that $m \geq 2$.*

Proof.

We use a reduction from the **SUBSET SUM** problem. Let M be a large constant such that $M > 3(\sum_{i=1}^{n} s_i + K)$. With the instance of **SUBSET SUM** problem we associate the following hyperbolic 0–1 programming problem:

$$\max_{\mathbf{x} \in \mathbb{B}^{n+1}} f(\mathbf{x}) = -\frac{2M}{M^2 - (2(\sum_{i=1}^{n} s_i x_i - K x_{n+1}) + 1 - x_{n+1})^2}. \tag{15}$$

If $x_{n+1} = 0$ then

$$f(\mathbf{x}) = -\frac{2M}{M^2 - (2\sum_{i=1}^{n} s_i x_i + 1)^2}. \tag{16}$$

Obviously, the maximum value of $f(\mathbf{x})$ will be $-2M/(M^2 - 1)$ if we have $x_1 = 0, x_2 = 0, \ldots, x_n = 0$. If $x_{n+1} = 1$ then

$$f(\mathbf{x}) = -\frac{2M}{M^2 - 4(\sum_{i=1}^{n} s_i x_i - K)^2}. \tag{17}$$

It is easy to observe that the **SUBSET SUM** problem has a solution if and only if $\max_{\mathbf{x}\in\mathbb{B}^{n+1}} f(\mathbf{x}) = -2/M$. Otherwise, $\mathbf{x} = (0,\ldots,0) \in \mathbb{B}^{n+1}$ is the global solution of (15) and $\max_{\mathbf{x}\in\mathbb{B}^{n+1}} f(\mathbf{x}) = -2M/(M^2-1)$. Therefore, the **SUBSET SUM** problem is reduced to checking if $\mathbf{x} = (0,\ldots,0) \in \mathbb{B}^{n+1}$ is the global solution of problem (15).

□

A similar result can also be proved for the single-ratio problem (2) applying the reduction described in [20] (see Lemma 4).

Although the above considered complexity results characterize worst-case instances, they provide some insight into the problem difficulty and indicate that for solving large scale problems we need to use some heuristics approaches.

3. Cardinality constrained problem

The *cardinality constrained hyperbolic 0–1 programming problem* is of the form:

$$\max_{\mathbf{x}\in\mathbb{B}^n} f(\mathbf{x}) = \sum_{j=1}^{m} \frac{a_{j0} + \sum_{i=1}^{n} a_{ji}x_i}{b_{j0} + \sum_{i=1}^{n} b_{ji}x_i}, \text{ s.t. } \sum_{i=1}^{n} x_i = p, \tag{18}$$

where constraint $\sum_{i=1}^{n} x_i = p$ is usually referred to as a *cardinality*, or *knapsack constraint*.

Problems of this type appear in scheduling common carriers [22] and p-choice facility location [26].

Let us recall the following definitions. We say that problem P is "polynomially reducible" to problem P_1 if given an instance $I(P)$ of problem P, we can in polynomial time obtain an instance $I(P_1)$ of problem P_1 such that solving $I(P_1)$ will solve $I(P)$. Two problems P_1 and P_2 are called "equivalent" if P_1 is "polynomially reducible" to P_2 and P_2 is "polynomially reducible" to P_1.

For quadratic 0–1 programming problem, which is probably the most known classical nonlinear 0–1 programming problem, it can be easily proved that cardinality constrained version of the problem is "equivalent" to the unconstrained one (see, for example, [10]). Next we show a similar result for our problem, i.e. if we require *only* condition (9) to be satisfied, the problems (1) and (18) are also "equivalent".

Proposition 1 *Problem (1) is "polynomially reducible" to problem (18).*

Proof. In order to optimize (1) we can solve $n+1$ problems (18) for each $p = 0,\ldots,n$. The optimum for problem (1) will be the maximum from the obtained results.

□

This result implies that any algorithm for solving cardinality constrained hyperbolic program (18) can be used as a procedure for solving unconstrained hyperbolic programs (1). Therefore, negative results on inapproximability of the problem (1) are also valid for the problem (18).

Proposition 2 *Problem (18) is "polynomially reducible" to problem (1).*

Proof. Without loss of generality we may assume that all coefficients a_{ji} and b_{ji} in the objective function of (18) are integers.

Reduction 1. Next define the following problem with $m+1$ ratios:

$$\max_{\mathbf{x}\in\mathbb{B}^n} g(\mathbf{x}) = \sum_{j=1}^{m} \frac{a_{j0} + \sum_{i=1}^{n} a_{ji}x_i}{b_{j0} + \sum_{i=1}^{n} b_{ji}x_i} - M \cdot \frac{\sum_{i=1}^{n} x_i - p}{2(\sum_{i=1}^{n} x_i - p) + 1}, \tag{19}$$

where $M > 6\sum_{j=1}^{m}\sum_{i=0}^{n}|a_{ji}|$. It is easy to check that if $\sum_{i=1}^{n} x_i \neq p$ then

$$\frac{\sum_{i=1}^{n} x_i - p}{2(\sum_{i=1}^{n} x_i - p) + 1} \geq \frac{1}{3}, \tag{20}$$

otherwise, this additional ratio is equal to 0. By the selection of M and (20) it is obvious that if $\sum_{i=1}^{n} x_i \neq p$ then $g(\mathbf{x}) < -\sum_{j=1}^{m}\sum_{i=0}^{n}|a_{ji}|$. Otherwise, $g(\mathbf{x}) \geq -\sum_{j=1}^{m}\sum_{i=0}^{n}|a_{ji}|$. Therefore, problem (19) is maximized *iff* $\sum_{i=1}^{n} x_i = p$, and $\max f(\mathbf{x}) = \max g(\mathbf{x})$. This reduction implies that problem (18) with m ratios can be reduced to problem (1) with $m+1$ ratios.

Reduction 2. Next define the following problem with m ratios:

$$\max_{\mathbf{x}\in\mathbb{B}^n} g(\mathbf{x}) = \sum_{j=1}^{m} \frac{a_{j0} + \sum_{i=1}^{n} a_{ji}x_i + 4M_jB_j(\sum_{i=1}^{n} x_i - p)}{b_{j0} + \sum_{i=1}^{n} b_{ji}x_i + 2B_j(p - \sum_{i=1}^{n} x_i)}, \tag{21}$$

where $M_j > \sum_{i=0}^{n}|a_{ji}|$ and $B_j > \sum_{i=0}^{n}|b_{ji}|$. It is easy to check that if $\sum_{i=1}^{n} x_i \neq p$ then each ratio is negative and $g(\mathbf{x}) < -\sum_{j=1}^{m}\sum_{i=0}^{n}|a_{ji}|$. Therefore, problem (21) is maximized *iff* $\sum_{i=1}^{n} x_i = p$, and $\max f(\mathbf{x}) = \max g(\mathbf{x})$. Problem (18) with m ratios can be reduced to problem (1) with the same number of ratios.

□

From **Proposition 1** and **Proposition 2** the following theorem follows:

Theorem 8 *Problems (1) and (18) are "equivalent".*

4. Linearization techniques

In this section we discuss linear mixed 0–1 reformulations of multiple-ratio hyperbolic 0–1 programming problems. We also assume that condition (9) is satisfied.

Li's and Wu's approaches and their modifications. Wu's linearization technique [28], which is an extension of Li's approach [15], is based on a very simple idea:

Theorem 9 *[28] A polynomial mixed 0–1 term $z = xy$, where x is a 0–1 variable, and y is a continuous variable taking any positive value, can be represented by the following linear inequalities: (1) $y - z \leq K - Kx$; (2) $z \leq y$; (3) $z \leq Kx$; (4) $z \geq 0$, where K is a large number greater than y.*

This result can be easily generalized for a general y, which is bounded by some lower and upper bounds (see [26]):

Corollary 1 *A polynomial mixed 0–1 term $z = xy$, where x is a 0–1 variable, and y is a continuous variable, can be represented by the following linear inequalities: (1) $z \leq Ux$; (2) $z \leq y + L(x-1)$; (3) $z \geq y + U(x-1)$; (4) $z \geq Lx$, where U and L are upper and lower bounds of variable y, i.e. $L \leq y \leq U$.*

Let

$$y_j = 1/(b_{j0} + \sum_{i=1}^{n} b_{ji}x_i). \tag{22}$$

It is assumed that condition (9) is satisfied. Then problem (1) becomes:

$$\max_{\mathbf{x}\in\mathbb{B}^n, \mathbf{y}\in\mathbb{R}^m} f(\mathbf{x},\mathbf{y}) = \sum_{j=1}^{m}(a_{j0}y_j + \sum_{i=1}^{n} a_{ji}x_iy_j), \tag{23}$$

$$\text{s.t. } b_{j0}y_j + \sum_{i=1}^{n} b_{ji}x_iy_j = 1,\ j = 1,\dots,m, \tag{24}$$

where objective function (23) is obtained replacing each term $1/(b_{j0}+\sum_{i=1}^{n} b_{ji}x_i)$ in (1) by y_j, and condition (24) is equivalent to (22) since (9) is satisfied.

Nonlinear terms x_iy_j in (23)-(24) can be linearized introducing new variable $z_{ij} = x_iy_j$ and applying **Theorem 9** (if condition (8) is satisfied), or **Corollary 1** (in general case).

Another possible reformulation can be constructed applying the following equality:

$$y_j = \frac{a_{j0} + \sum_{i=1}^{n} a_{ji}x_i}{b_{j0} + \sum_{i=1}^{n} b_{ji}x_i}.$$

In this case problem (1) is reformulated as:

$$\max_{\mathbf{x}\in\mathbb{B}^n, \mathbf{y}\in\mathbb{R}^m} f(\mathbf{x},\mathbf{y}) = \sum_{j=1}^{m} y_j, \tag{25}$$

$$\text{s.t. } b_{j0}y_j + \sum_{i=1}^{n} b_{ji}x_iy_j = a_{j0} + \sum_{i=1}^{n} a_{ji}x_i,\ j = 1,\dots,m. \tag{26}$$

Nonlinear terms x_iy_j in (26) should be linearized using **Corollary 1**.

The number of new variables z_{ij} in both formulations is $m + mn$.

In more details formulations (23)-(24) and (25)-(26), their modifications and some other aspects of linearization techniques (estimation of bounds on the

fractional terms, additional constraints for tighter relaxations) are discussed in [15, 26, 27, 28].

New modification. The following theorem can be formulated as a generalization of **Theorem 9**:

Theorem 10 *A polynomial mixed 0-1 term $z = y(c_1x_1 + c_2x_2)$, where x_1, x_2 are 0-1 variables, c_1 and c_2 are some positive constant numbers and y is a continuous variable taking any positive value, can be represented by the following linear inequalities: (1) $z \geq 0$, (2) $z \leq K(c_1x_1 + c_2x_2)$, (3) $z \leq c_1y + c_2y$, (4) $z \leq c_1y + Kc_2x_2$, (5) $z \leq c_2y + Kc_1x_1$, (6) $z \geq c_1y - Kc_1(1 - x_1)$, (7) $z \geq c_2y - Kc_2(1 - x_2)$, (8) $z \geq c_1y + c_2y - Kc_1(1 - x_1) - Kc_2(1 - x_2)$, where K is a large number greater than y.*

Proof.

We need to check the following four variants: (1) $x_1 = 0$ and $x_2 = 0$; (2) $x_1 = 1$ and $x_2 = 0$; (3) $x_1 = 0$ and $x_2 = 1$; (4) $x_1 = 1$ and $x_2 = 1$.

- If $x_1 = 0$ and $x_2 = 0$ then term z must be equal to 0. Conditions (1) and (2) force $z = 0$. Conditions (3)-(5) are satisfied since K, y, c_1, and c_2 are nonnegative. In (6) we have that $c_1y - Kc_1(1 - x_1) = c_1y - Kc_1 = c_1(y - K) \leq 0$. Since $z = 0$ (6) is obviously satisfied. Similarly, conditions (7) and (8) are also valid.
- If $x_1 = 1$ and $x_2 = 0$ then term z must be equal to c_1x_1. Conditions (4) and (6) force $z = c_1y_1$. It is easy to check that the rest of the inequalities are satisfied.
- The last two variants can be checked similarly.

□

Let

$$y_j = \frac{a_{j0} + \sum_{i=1}^{n} a_{ji}x_i}{b_{j0} + \sum_{i=1}^{n} b_{ji}x_i} + M_j,$$

where M_j is a constant large enough such that $y_j \geq 0$. The obtained reformulation is similar to (25)-(26):

$$\max_{\mathbf{x}\in\mathbb{B}^n, \mathbf{y}\in\mathbb{R}^m} f(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{m} y_j, \tag{27}$$

$$\text{s.t. } b_{j0}y_j + \sum_{i=1}^{n} b_{ji}x_iy_j = a_{j0} + \sum_{i=1}^{n} a_{ji}x_i + M_j(b_{j0} + \sum_{i=1}^{n} b_{ji}x_i),\ j = 1, \ldots, m. \tag{28}$$

Nonlinear terms x_iy_j should be linearized using the approach described in **Theorem 10**. The advantage of the proposed modification (we have 2 binary variables corresponding to each new variable, see **Theorem 10**) is that the number of new variables is at most $m + m(\lfloor n/2 \rfloor + 1) \sim m + mn/2$ while the number of constraints *remains the same.* Note also that we can formulate theorems similar to **Theorem 10**, where $z = y(c_1x_1 + c_2x_2 + c_3x_3)$,

$z = y(c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4)$, etc. Applying these reformulations we obtain new linear mixed 0–1 formulations, but in this case more constraints should be generated (actually the number of constraints grows exponentially).

Example. Let us illustrate the proposed linearization with the following example. Suppose we need to linearize the following problem:

$$\min_{\mathbf{x}\in\mathbb{B}^n} \frac{1 + x_1}{8 + x_1 + 2x_2 - 3x_3 - 4x_4} \tag{29}$$

Let $y = (1 + x_1)/(8 + x_1 + 2x_2 - 3x_3 - 4x_4)$. Obviously, $0 < y \leq 1$ and the above formulation then becomes:

$$\begin{array}{c} \min y, \\ \text{s.t.} \quad 8y + yx_1 + 2yx_2 - 3yx_3 - 4yx_4 = 1 + x_1, \\ x_1,\ x_2 \in \{0, 1\} \end{array} \tag{30}$$

Applying a standard technique we need to introduce 4 new variables z_i for each term $z_i = yx_i$ $(i = 1, \ldots, 4)$ plus 16 additional inequalities. In a new approach we need only 2 variables z_i such that $z_1 = y(x_1 + 2x_2)$ and $z_2 = y(3x_3 + 4x_4)$, and the same number of inequalities.

5. GRASP-based heuristic

The complexity results considered in Section 2 of this paper indicate that hyperbolic 0–1 programming problem is a difficult combinatorial optimization problems, and, therefore, in order to obtain good quality solution in a reasonable amount of time we need to use some heuristics approaches. In this section we propose a metaheuristic algorithm for solving the cardinality constrained multiple-ratio hyperbolic 0–1 programming problem (18). The algorithm we present is based on a *Greedy Randomized Adaptive Search Procedure (GRASP)*. To the authors' best knowledge no results on applying GRASP-based heuristic approaches for the considered problem have been reported yet. Although the algorithm described below is rather simple we were able to obtain good results.

GRASP is a sampling procedure, proposed by Feo and Resende [5], which tries to create good solutions with high probability. The main tools to make this possible are the construction and the improvement phases. In the construction phase, GRASP creates a complete solution by iteratively adding components of a solution with the help of a greedy function, used to perform the selection. In the case of the hyperbolic function problem, a solution is composed by a set of 0–1 variables. Thus, a solution is created by defining for each individual variable a value in $\{0, 1\}$.

The improvement phase then takes the incumbent solution and performs local perturbations in order to get a local optimal solution, with respect to some predefined neighborhood. Different local search algorithms can be defined according to the neighborhood chosen. The general GRASP procedure can be described as in Algorithm 1.

```
initialize variables;
while termination criterion not satisfied do
    /* Construction phase */ ;
    s ← ∅ ;
    while solution s not feasible do
        Order available components according to greedy function g ;
        Select one (say y) of the best α components /* α is a parameter
        */ ;
        Add component to solution s: s ← s ∪ {y} ;
    end
    /* Local search phase */ ;
    while local search criteria not satisfied do
        Perform local perturbation on s ;
        if solution s improved then
            Keep changes ;
        end
    end
    Save best solution ;
end
```

Algorithm 1: GRASP

Problem Formulation. More specifically the problem we consider in this section is of the form:

$$\max_{\mathbf{x}\in\mathbb{B}^n} f(\mathbf{x}) = \sum_{j=1}^{m} \frac{a_{j0} + \sum_{i=1}^{n} a_{ji}x_i}{b_{j0} + \sum_{i=1}^{n} b_{ji}x_i}, \tag{31}$$

$$\text{s.t.} \ \sum_{i=1}^{n} x_i = p, \tag{32}$$

$$b_{j0} + \sum_{i=1}^{n} b_{ji}x_i > 0 \ \text{ for } j = 1, \ldots, m \tag{33}$$

The reason for adding the last constraint is the following. In many of the cases, where we consider multiple-ratio hyperbolic 0–1 programming problems, condition (8) is either satisfied, or enforced by adding the set of constraints (33) to the problem formulation (for example, see [26, 28]).

Construction Phase. The construction phase of our GRASP consists of defining an initial assignment of the values 0 or 1 to each one of the variables. The component (variable) that will be assigned at each iteration is chosen according to the amount of its contribution to the solution. This means that the greedy approach used, tries to maximize the partial objective function corresponding to the values already assigned. The partial objective function f' can

be described in the following way

$$f'(x) = \sum_{j=1}^{m} \left(\frac{\sum_{i \in S'} a_{ji} x_i}{\sum_{i \in S'} b_{ji} x_i} - \frac{\sum_{i \in S} a_{ji} x_i}{\sum_{i \in S} b_{ji} x_i} \right), \tag{34}$$

where S is the original set of selected indices and S' is the new set of indices after the definition of the value of one additional variable in solution x. The GRASP

---

input: for every x_i the corresponding a_{ji}, b_{ji} for all j and p ;
output: a vector x for the hyperbolic function, with $x \in \{0,1\}^n$;
/* initialize solution x */ ;
$x \leftarrow (0, \ldots, 0)$;
$S \leftarrow \emptyset$; $\bar{S} \leftarrow \emptyset$;
for $i \leftarrow 1$ *to* n **do**
  /* create a restricted candidate list l */ ;
  $L \leftarrow \{1, \ldots, n\} \setminus (S \cup \bar{S})$;
  Order L according to function f' as described in equation (34) ;
  RCL $\leftarrow$ first α elements of L ;
  Select random index $i \in$ RCL ;
  $x_i \leftarrow 1$;
  if *there is any denominator* < 0 **then**
    set $x_i \leftarrow 0$ and $\bar{S} \leftarrow \bar{S} \cup \{i\}$;
  else
    $S \leftarrow S \cup \{i\}$;
    if $\sum_{i=1}^{n} x_i = p$ **then return** x ;
  end
end

Algorithm 2: Construction phase

algorithm uses a list of candidate components, also known as the restricted candidate list (RCL). In our case, the RCL is composed of the α best indices according to equation (34). Therefore, during the construction phase we sort the candidate variables in decreasing order according to its marginal contribution ($f'(x)$) to the objective function.

The implementation details of the construction phase are presented in Algorithm 2. During the procedure, two sets of indices are maintained:

- the set S of indices of variables that have been selected and assigned the value 1;
- the set $\bar{S}$ of indices of variables that have been defined as infeasible (one of the denominators is negative) for the current solution (and therefore will have value 0).

Variables are included in S whenever they are selected from the RCL, and receive a value 1. On the other hand, if a variable is found to be infeasible for the

current solution, its index is included in $\bar{S}$. Parameter α is a random variable, uniformly distributed between 1 and the size of the list for each iteration of the Algorithm 2.

Note that due to the nature of the random choices made in the construction phase, it is possible that a particular sequence of chosen variables lead to an infeasible solution. This is handled in the algorithm by simply discarding the infeasible solution and re-starting the construction phase.

Handling Constraints in GRASP. An important part of solving the hyperbolic function problem is handling the feasibility of generated solutions. A method to handle the linear constraints is to guarantee from the beginning that only feasible solutions are generated. This can be made possible by carefully checking each candidate solution, and making sure that all the constraints are satisfied. In our algorithm, a feasibility checking function is applied each time a new solution is considered for the problem and, therefore, we avoid problems created by infeasible solutions. During the construction phase, when solving knapsack instances of the problem, we only test if the current solution has denominators greater than zero, since a partial solution with $\sum_{i=1}^{n} x_i < p$ can become feasible in the next iterations.

Improvement Phase. The improvement phase of GRASP has the objective of finding a local optimal solution according to a local neighborhood. The neighborhood of our problem is defined by perturbations on the incumbent solution. The perturbation consists of selecting two variables x_i and x_j such that $x_i \neq x_j$ and flipping their values to zero or one, while keeping $\sum_{i=1}^{n} x_i = p$. The variables are selected randomly, and after a change of values in the selected variables is performed, the resulting solution is tested for feasibility (the denominators must remain positive). If feasibility is achieved, then the solution is accepted if its cost is better than the previous one. Otherwise, a new random perturbation of the solution is done. This phase ends after N iterations without improvement, where N is a parameter of the algorithm. In the computational experiments reported below $N = 1000$.

The formal procedure is described in Algorithm 3.

Computational Results. The algorithm described above was implemented using the C language and compiled with the `gcc` compiler. The tests were performed in a machine with the Intel Pentium 4 CPU at 2.7GHz. The operating system used was Windows XP.

Test instances were constructed using the following idea. All coefficients a_{ji} and b_{ji} are integers randomly generated from the interval [-100,100] (see Table 1), or [1,100] (see Table 2).

Since all coefficients a_{ji} and b_{ji} are integers, constraints (33) are replaced by equivalent constraints of the form:

$$b_{j0} + \sum_{i=1}^{n} b_{ji} x_i \geq 1 \qquad \text{for } j = 1, \ldots, m \tag{35}$$

input: for every x_i the corresponding a_{ji}, b_{ji} for all j and p; the current solution $x \in \{0,1\}^n$;
output: a local maximum x for the hyperbolic function f, with $x \in \{0,1\}^n$;
$k \leftarrow 0$;
while $k < N$ **do**
    /* perturb solution */ ;
    Select two random indexes i and j, such that $i \neq j$ and $x_i \neq x_j$;
    $x' \leftarrow x$; $x'_i \leftarrow 1 - x'_i$; $x'_j \leftarrow 1 - x'_j$;
    $c \leftarrow f(x')$;
    /* save perturbed solution if necessary */ ;
    if $c > f(x)$ *and* x' *is feasible* **then**
        $x_i \leftarrow 1 - x_i$;
        $x_j \leftarrow 1 - x_j$;
        $k \leftarrow 0$;
    end
    $k \leftarrow k + 1$;
end

Algorithm 3: Improvement phase

In the 2nd class of the test problems instead of maximization we considered minimization problem.

Tables 1 and 2 summarize results found with the proposed algorithm. These tables are organized as follows. The first four columns give information about the instances: the number of variables (n), the number of ratios (m), the number of elements in the knapsack constraint (k), and the random seed used by the generator (which is publicly available). The next four columns present the results of the exact algorithm used, in comparison to GRASP.

For the exact algorithm Wu's linearization (23)-(24) was used. Since all generated coefficients are integers all fractional terms can be upper bounded by K=1 (see **Theorem 9**).

The integer program solver was CPLEX 9.0 [29].

In both cases the CPU time (in seconds) and the value of the best solution found are reported. The time reported for GRASP is for the iteration where the best solution was found by the algorithm.

The termination criterion for GRASP is the following. The algorithm is set up to run while a fixed number of iterations is reached without any improvement. In the tests presented above this number was set to $10,000$. However, in most cases the best solution is found with just a few iterations, as can be seen from the small time needed to find the optimum solution.

instance				exact		GRASP	
n	m	k	seed	time(s)	value	time(s)	value
20	10	10	535	39.44	617.84	1.75	617.84
20	10	10	756	18.42	416.88	6.27	416.88
20	10	10	846	69.83	518.64	1.17	518.64
20	10	10	856	34.75	453.62	0.08	453.62
20	10	15	136	9.95	770.33	4.68	770.33
20	10	15	674	26.48	469.98	47.07	469.98
20	10	15	756	6.80	335.93	0.80	335.93
20	10	15	757	7.30	416.94	0.91	416.94
20	10	15	876	8.27	477.25	0.01	477.25
25	10	10	565	700.26	726.54	37.56	726.54
25	10	10	754	129.58	747.55	1.50	747.55
25	10	10	755	185.66	431.41	0.27	431.41
25	10	10	756	22.33	476.70	1.81	476.70
25	10	10	855	100.53	744.20	0.64	744.20
25	10	15	733	507.39	797.60	1.31	797.60
25	10	15	743	1782.69	680.96	108.73	680.96
25	10	15	744	99.05	872.78	3.45	872.78
25	10	15	754	109.08	855.28	0.88	763.12
25	10	15	865	201.45	464.25	1.80	464.25
25	2	10	565	0.78	536.21	0.03	536.21
25	2	10	754	0.86	620.37	1.24	620.37
25	2	10	755	0.42	194.37	0.25	194.37
25	2	10	777	0.23	609.14	2.23	609.14
25	2	15	733	0.30	745.90	0.91	745.90
25	2	15	743	0.48	605.95	0.80	605.95
25	2	15	744	1.25	866.42	4.36	866.42
25	2	15	754	0.36	675.47	0.69	675.47
25	2	15	865	0.33	308.12	0.14	308.12

Table 1: Results to instances with $a_{ji}, b_{ji} \in [-100, 100]$.

instance				exact		GRASP	
n	m	k	seed	time(s)	value	time(s)	value
40	2	10	5653	1.53	0.24	0.00	0.24
40	2	10	7567	0.83	0.25	0.00	0.25
40	2	10	8464	0.42	0.19	0.02	0.19
40	2	10	8767	0.28	0.16	0.00	0.16
40	2	15	1667	6.34	0.30	0.02	0.30
40	2	15	6643	2.17	0.33	0.02	0.33
40	2	15	7545	20.73	0.36	0.05	0.36
40	2	15	7546	4.73	0.33	0.00	0.33
40	2	15	8754	10.41	0.35	0.02	0.35
40	2	20	7435	102.88	0.40	0.00	0.40
40	2	20	7534	9.36	0.47	0.00	0.47
40	2	20	8434	579.66	0.50	0.02	0.50
40	2	20	8534	2.69	0.38	0.02	0.38
40	2	25	5443	47.38	0.61	0.02	0.61
40	2	25	6443	176.28	0.69	0.02	0.69
40	2	25	8444	6.42	0.63	0.02	0.63
40	2	25	8544	21.56	0.65	0.03	0.65
45	2	10	5674	0.92	0.19	0.03	0.19
45	2	10	7573	0.70	0.19	0.03	0.19
45	2	10	7574	0.27	0.16	0.02	0.16
45	2	10	7575	1.26	0.20	0.00	0.20
45	2	10	8564	5.84	0.28	0.02	0.28
45	2	15	7395	16.28	0.34	0.03	0.34
45	2	15	7493	2.30	0.29	0.02	0.29
45	2	15	7494	21.58	0.40	0.06	0.40
45	2	15	7594	53.48	0.36	0.02	0.36
45	2	15	8694	2.25	0.30	0.06	0.30
45	2	20	4393	308.30	0.39	0.02	0.39
45	2	20	5575	90.23	0.40	0.01	0.40
45	2	20	6686	51.20	0.40	0.02	0.40
45	2	20	7463	23.81	0.40	0.02	0.40
45	2	20	7767	11.42	0.38	0.00	0.38
45	2	25	7453	445.05	0.52	0.02	0.52
45	2	25	7456	302.45	0.57	0.03	0.57
45	2	25	7643	31.95	0.48	0.03	0.48
45	2	25	7653	683.17	0.52	0.03	0.52
45	2	25	7656	517.70	0.50	0.02	0.50

Table 2: Results to instances with $a_{ji}, b_{ji} \in [1, 100]$.

6. Concluding Remarks

In this paper, we have discussed multiple-ratio hyperbolic 0–1 (fractional 0–1) programming problems. We have investigated some theoretical aspects of these problems including complexity issues, cardinality constrained cases and linear mixed 0–1 reformulations. Although the considered complexity results indicate that the multiple-ratio hyperbolic 0–1 programming problem is extremely difficult, the proposed GRASP-based heuristic for solving cardinality constrained problems has proved to perform very well.

References

[1] D. Armstrong, S. Jacobson, Studying the complexity of global verification for NP-hard discrete optimization problems, Journal of Global Optimization, 27, pp. 83-96, 2003.

[2] E. Boros, P. Hammer, Pseudo-Boolean Optimization. Discrete Applied Mathematics, 123(1-3), pp. 155–225, 2002.

[3] A. Charnes, W.W. Cooper, Programming with linear fractional functionals, Naval Research Logistics Quarterly 9 (1962), pp. 181–186.

[4] U. Feige, A threshold of $\ln n$ for approximating set cover. Journal of the ACM, 45(4), pp. 634–652, 1998.

[5] T.A. Feo, M.G.C. Resende, Greedy randomized adaptive search procedures, J. of Global Optimization, 6, pp. 109–133, 1995.

[6] R. Freund, F. Jarre, Solving the Sum-of-Ratios Problem by an Interior-Point Method, Journal of Global Optimization, 19, pp. 83-102, 2001.

[7] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, San Francisco, 1979.

[8] P. Hansen, M. Poggi de Aragão, C.C. Ribeiro, Hyperbolic 0–1 programming and query optimization in information retrieval, Mathematical Programming, 52, pp. 256–263, 1991.

[9] S. Hashizume, M. Fukushima, N. Katoh, T. Ibaraki, Approximation algortihms for combinatorial fractional programming problems, Mathematical Programming, 37, pp. 255-267.

[10] L.D. Iasemidis, P.M. Pardalos, J.C. Sackellares, and D.S. Shiau, Quadratic binary programming and dynamical system approach to determine the predictability of epileptic seizures, Journal of Combinatorial Optimization, 5 (1):9–26, 2001.

[11] S.H. Jacobson, D. Solow, The effectiveness finite improvement algorithms for finding global optima, Methods and Models of Operations Research, 37, pp. 257-272, 1993.

[12] D.S. Johnson, C.H. Papadimitriou, M. Yannakakis, How easy is local search?, J. Comput. System Sci. 37/1. pp. 79–1000, 1988.

[13] H. Konno, K. Fukaishi, A branch and bound algorithm for solving low rank linear multiplicative and fractional programming problems, Journal of Global Optimization, 18, pp. 283-299, 2000.

[14] T. Kuno, A branch and bound algorithm for maximizing the sum of several linear ratios, Journal of Global Optimization, 22, pp. 155-174, 2002.

[15] H. Li, A global approach for general 0–1 fractional programming, European J. of Operations Research, 73, pp. 590–596, 1994.

[16] C. Lund, M. Yannakakis, On the hardness of approximating minimization problem, JACM 41(5), 1994, pp. 960–981.

[17] C.H. Papadimitriou, K. Steiglitz, On the complexity of local search for the travelling salesman problem, SIAM Journal on Computing, 6, pp. 76-83, 1988.

[18] P.M. Pardalos, S. Jha, Complexity of uniqueness and local search in quadratic 0-1 programming, Operations Research Letters, Vol. 11, pp. 119–123, 1992.

[19] J.-C. Picard and M. Queyranne, A network flow solution to some nonlinear 0-1 programming problems, with applications to graph theory, Networks, 12 (1982), pp. 141-159.

[20] O. A. Prokopyev, H.-X. Huang, P.M. Pardalos, On complexity of unconstrained hyperbolic 0-1 programming problems, Operations Research Letters, Vol. 33/3 (2005), pp. 312–318.

[21] I. Quesada, I.E. Grossman, A global optimization algorithm for linear fractional and bilinear programs, Journal of Global Optimization, 6, pp. 39-76, 1995.

[22] S. Saipe, Solving a (0,1) hyperbolic program by branch and bound, Naval Research Logistics Quarterly, 22, pp. 497–515, 1975.

[23] S. Schaible, Fractional Programming, In: R. Horst and P.M. Pardalos (eds.), Handbook of Global Optimization, Kluwer Academic Publishers, Norwell, pp. 495-608, 1995.

[24] S. Schaible, Fractional Programming: the sum-of-ratios case, Optimization Methods and Software, Vol. 18, No. 2, pp. 219-229, 2003.

[25] I.M. Stancu-Minasian, Fractional Programming, Kluwer Academic Publishers, Dordrecht, 1997.

[26] M. Tawarmalani, S. Ahmed, N. Sahinidis, Global Optimization of 0–1 Hyperbolic Programs, Journal of Global Optimization, 24, pp. 385–416, 2002.

[27] M. Tawarmalani, S. Ahmed, N. Sahinidis, Reformulations of Rational Functions of 0-1 Variables, manuscript (http://archimedes.scs.uiuc.edu/papers/rational.pdf).

[28] T.-H. Wu, A note on a global approach for general 0–1 fractional programming, European J. of Operations Research, 101, pp. 220–223, 1997.

[29] ILOG Inc. CPLEX 9.0 User's Manual, 2004.