RAIRO Operations Research
RAIRO Oper. Res. **40** (2006) 97–111
DOI: 10.1051/ro:2006013

RÉSOLUTION D'UN PROBLÈME COMBINATOIRE FRACTIONNAIRE PAR LA PROGRAMMATION LINÉAIRE MIXTE

Alain Billionnet[1] et Karima Djebali[1]

Abstract. Fractionnal mathematical programs appear in numerous operations research, computer science and economic domains. We consider in this paper the problem of maximizing the sum of 0–1 hyperbolic ratios (SRH). In contrast to the single ratio problem, there has been little work in the literature concerning this problem. We propose two mixed-integer linear programming formulations of SRH and develop two different strategies to solve them. The first one consists in using directly a general-purpose mixed-integer programming solver. The second one is based on a specialized branch and bound algorithm that reformulates more precisely the problem at every node of search tree. We also propose a heuristic method and we exploit the obtained solution in order to improve the first strategy. We present computational experiments that allow to compare the different approaches.

Résumé. Les programmes mathématiques fractionnaires apparaissent dans de nombreux domaines de la recherche opérationnelle, de l'informatique et de l'économie. Nous considérons dans ce papier le problème de la maximisation de la somme de plusieurs ratios hyperboliques en variables 0-1 (SRH). Contrairement à la maximisation d'un seul ratio, très peu d'auteurs se sont intéressés à ce problème. Nous proposons deux formulations de SRH par des programmes linéaires en variables mixtes et deux stratégies différentes pour résoudre ces programmes. La première consiste à appliquer directement un outil général de programmation linéaire en variables mixtes et la deuxième est fondée sur un algorithme de branch and bound spécifique qui utilise une réécriture plus précise des programmes en chaque noud de l'arbre de recherche. Nous proposons également une résolution de SRH par une méthode heiristique et nous exploitons la solution ainsi obtenue pour améliorer la première stratégie. Nous présentons des résultats expérimentaux permettant de comparer les différentes approches.

Reçu le 31 décembre, 2005. Accepté le 31 décembre, 2005.

[1] CEDRIC-IIE, 18 allée Jean Rostand, 91025 Evry Cedex, France ;
{billionnet,djebali}@iie.cnam.fr

Mots Clés. Optimisation combinatoire fractionnaire, somme de ratios hyperboliques, programmation linéaire en variables mixtes, branch-and-bound, expérimentation.

1. Introduction

Les programmes mathématiques fractionnaires, c'est-à-dire les programmes dont l'objectif s'exprime comme le rapport de deux fonctions, en variables réelles, en variables entières ou en variables 0-1 apparaissent dans plusieurs domaines tels que les bases de données, l'optimisation combinatoire, la programmation stochastique et l'économie (voir, par exemple, [4, 5, 7, 8, 12–14, 16, 17, 19]). Nous considérons dans cet article le problème en variables 0-1 qui consiste à optimiser la somme de plusieurs ratios hyperboliques. Il s'écrit comme suit :

$$(SRH) \ : \ \text{Max} \left\{ f(x) = \sum_{i=1}^{m} \frac{a_i + \sum_{j=1}^{n} a_{ij} x_j}{b_i + \sum_{j=1}^{n} b_{ij} x_j} \ : \ x \in \{0,\, 1\}^n \right\}$$

où $a_i \geq 0$, $b_i > 0$ $(i = 1, ..., m)$, $a_{ij} \geq 0$, $b_{ij} \geq 0$ $(i = 1, ..., m\,;\, j = 1, ..., n)$. Posons $N = \{1, ..., n\}$ et $M = \{1, ..., m\}$.
Notons que pour le problème à un seul ratio, $\text{Max} \left\{ \frac{a_1 + \sum_{j=1}^{n} a_{1j} x_j}{b_1 + \sum_{j=1}^{n} b_{1j} x_j} \ : \ x \in S \subseteq \mathbb{R}^n \right\}$ où S est une région convexe, un maximum local est un maximum global. Ce n'est plus le cas pour le problème à plusieurs ratios, $\text{Max}\{f(x) \ : \ x \in S \subseteq \mathbb{R}^n\}$.

Diverses applications peuvent se modéliser sous la forme SRH (voir, par exemple, [1, 6, 7, 15]). Contrairement à la maximisation d'un seul ratio hyperbolique, peu d'études concernent ce problème. Saipe [15] considère le problème SRH avec une contrainte de cardinalité et propose de le résoudre par un algorithme de branch-and-bound. Plus récemment, Li [11] et Wu [20] ont formulé le problème par un programme linéaire en variables mixtes (PLVM) et utilisent un outil standard de programmation linéaire pour sa résolution [18]. Cette technique de modélisation d'un programme non linéaire en variables 0-1 par un PLVM est bien connue et éprouvée mais il convient de la mettre en œuvre avec prudence car certaines formulations peuvent nécessiter des temps de calcul importants. Nous nous intéressons dans ce travail à la résolution de SRH par la PLVM. Nous présentons tout d'abord la formulation de Wu [20] et une amélioration de cette formulation puis une nouvelle formulation du problème par la PLVM. Pour résoudre les programmes obtenus, nous proposons deux stratégies. La première consiste à appliquer directement un outil général de programmation linéaire en variables mixtes et la deuxième est fondée sur un algorithme de branch-and-bound spécifique qui

utilise une réécriture plus précise des programmes en chaque nœud de l'arbre de recherche. Cette réécriture consiste à ajuster les coefficients intervenant dans l'étape de linéarisation. Nous proposons également une résolution de SRH par une méthode heuristique et nous exploitons la solution ainsi obtenue pour améliorer la première stratégie. Nous présentons des résultats expérimentaux permettant de comparer les différentes approches.

2. Deux reformulations linéaires du problème

Dans cette partie, nous présentons deux techniques de linéarisation qui permettent de formuler le problème SRH sous la forme d'un PLVM. Nous exposons tout d'abord la formulation de Wu [20] puis nous proposons une nouvelle formulation.

2.1. Linéarisation de Wu [20] ($LIN1$)

Proposée dans [20], cette linéarisation, qui pourrait s'appliquer au cas d'un seul ratio, consiste tout d'abord à remplacer l'inverse des dénominateurs de chaque ratio par les variables continues y_i ($y_i = 1/(b_i + \sum_{j=1}^{n} b_{ij}x_j)$, ($i = 1, ..., m$)) puis à linéariser le programme quadratique obtenu. On obtient ainsi le programme quadratique SRH' où la contrainte (1) exprime l'égalité $y_i = 1/(b_i + \sum_{j=1}^{n} b_{ij}x_j)$.

$$(SRH') \begin{cases} \text{Max} \sum_{i=1}^{m} \left(a_i y_i + \sum_{j=1}^{n} a_{ij} y_i x_j \right) & \\ b_i y_i + \sum_{j=1}^{n} b_{ij} y_i x_j = 1 & i \in M \quad (1) \\ x_j \in \{0, 1\} & j \in N. \end{cases}$$

Une façon classique de linéariser SRH' consiste à remplacer le produit $y_i x_j$, comportant une variable continue et une variable bivalente, par une seule variable z_{ij} tout en ajoutant des contraintes de linéarisation. Ces contraintes forcent la variable z_{ij} à prendre la valeur du produit $y_i x_j$.

Lemme 2.1. [20]. Pour tout $x \in \{0, 1\}$ et pour tout $y \in [0, K]$, $z = yx$ si et seulement si les contraintes (C) ci-dessous sont vérifiées.

$$(C) \begin{cases} y - z \leq K(1 - x) & (2) \\ z \leq y & (3) \\ z \leq Kx & (4) \\ z \geq 0. & (5) \end{cases}$$

La preuve est évidente en considérant successivement les deux valeurs possibles de la variable x.

Appliquons le lemme 2.1 au programme SRH', en remplaçant le produit $y_i x_j$ par z_{ij} pour tout $i \in M$ et pour tout $j \in N$. On obtient le théorème 2.2 où $\overline{y}_i$ désigne une borne supérieure de y_i dans toute solution optimale de SRH.

Théorème 2.2. *Soit* (x^*, y^*, z^*) *une solution optimale de* $LIN1$. x^* *est une solution optimale de* SRH.

$$(LIN1)\begin{cases} \text{Max} \sum_{i=1}^{m} \left(a_i y_i + \sum_{j=1}^{n} a_{ij} z_{ij} \right) & \\ b_i y_i + \sum_{j=1}^{n} b_{ij} z_{ij} = 1 & i \in M \\ z_{ij} \geq y_i - \overline{y}_i (1 - x_j) & i \in M,\ j \in N \\ z_{ij} \leq y_i & i \in M,\ j \in N \\ z_{ij} \leq \overline{y}_i x_j & i \in M,\ j \in N \\ x_j \in \{0, 1\},\ z_{ij} \geq 0 & i \in M,\ j \in N. \end{cases}$$

La preuve est une conséquence directe du lemme 2.1.

Remarquons que la linéarisation de SRH entraîne une augmentation importante du nombre de variables. En effet, le programme linéaire en variables mixtes $LIN1$ comporte n variables bivalentes et $m(n+1)$ variables réelles alors que SRH comporte seulement n variables bivalentes. De plus $LIN1$ comporte $m + 3mn$ contraintes alors que SRH est un problème sans contraintes.

2.2. Une nouvelle linéarisation ($LIN2$)

L'idée de cette linéarisation est d'écrire la fonction objectif comme la somme de m variables r_i, $i \in M$ où r_i est égale au i-ème ratio, c'est-à-dire, $r_i = \frac{a_i + \sum_{j=1}^{n} a_{ij} x_j}{b_i + \sum_{j=1}^{n} b_{ij} x_j}$. Cette technique de linéarisation est utilisée dans [2] pour le traitement du problème du sac à dos hyperbolique qui consiste à maximiser un seul ratio de fonctions linéaires sous une contrainte de capacité linéaire.

Le programme SRH devient alors le programme quadratique SRH'' où la contrainte (6) exprime l'égalité entre r_i et le i-ème ratio.

$$(SRH'')\begin{cases} \text{Max} \sum_{i=1}^{m} r_i & \\ a_i + \sum_{j=1}^{n} a_{ij} x_j = b_i r_i + \sum_{j=1}^{n} b_{ij} r_i x_j & i \in M \ (6) \\ x_j \in \{0, 1\} & j \in N. \end{cases}$$

En remplaçant le produit $r_i x_j$ par la variable z_{ij} et en appliquant le lemme 2.1, nous obtenons le PLVM $SRH_L^{''}$ ci-dessous :

$$(SRH_L^{''}) \begin{cases} \text{Max} \sum_{i=1}^{m} r_i & \\ a_i + \sum_{j=1}^{n} a_{ij} x_j = b_i r_i + \sum_{j=1}^{n} b_{ij} z_{ij} & i \in M \qquad (6^{'}) \\ z_{ij} \geq r_i - \overline{r}_i (1 - x_j) & i \in M,\, j \in N \quad (7) \\ z_{ij} \leq r_i & i \in M,\, j \in N \quad (8) \\ z_{ij} \leq \overline{r}_i x_j & i \in M,\, j \in N \quad (9) \\ x_j \in \{0,\, 1\},\, z_{ij} \geq 0 & i \in M,\, j \in N \quad (10) \end{cases}$$

où $\overline{r}_i$ désigne une borne supérieure de r_i, c'est-à-dire du i-ème ratio, dans toute solution optimale de SRH.

Remarquons que les contraintes (8)–(9) ne sont pas nécessaires pour forcer la variable z_{ij} a être égale au produit $r_i x_j$. On obtient ainsi le théorème suivant.

Théorème 2.3. *Soit* $(x^*,\, r^*,\, z^*)$ *une solution optimale de* $LIN2$. x^* *est une solution optimale de* SRH.

$$(LIN2) \begin{cases} \text{Max} \sum_{i=1}^{m} r_i & \\ a_i + \sum_{j=1}^{n} a_{ij} x_j = b_i r_i + \sum_{j=1}^{n} b_{ij} z_{ij} & i \in M \qquad (6^{'}) \\ z_{ij} \geq r_i - \overline{r}_i (1 - x_j) & i \in M,\, j \in N \ (7) \\ x_j \in \{0,\, 1\},\, z_{ij} \geq 0 & i \in M,\, j \in N \ (10). \end{cases}$$

Démonstration. Supposons que $\hat{x} \in \{0,\, 1\}^n$ soit meilleure que x^*, c'est-à-dire $f(\hat{x}) > f(x^*)$. Soit $(\hat{x},\, \hat{r},\, \hat{z})$ tel que $\hat{r}_i = \frac{a_i + \sum_{j \in N} a_{ij} \hat{x}_j}{b_i + \sum_{j \in N} b_{ij} \hat{x}_j}$ $(i \in M)$, $\hat{z}_{ij} = \hat{r}_i \hat{x}_j$ $(i \in M,\, j \in N)$.

i) Montrons que $(\hat{x},\, \hat{r},\, \hat{z})$ est admissible pour $LIN2$.
   $b_i \hat{r}_i + \sum_{j \in N} b_{ij} \hat{z}_{ij} = b_i \hat{r}_i + \sum_{j \in N} b_{ij} \hat{r}_i \hat{x}_j = \hat{r}_i (b_i + \sum_{j \in N} b_{ij} \hat{x}_j) = a_i + \sum_{j \in N} a_{ij} \hat{x}_j$ et (6') est donc vérifiée ;
   (10) est vérifiée ;
   si $\hat{x}_j = 1$ alors $\hat{z}_{ij} = \hat{r}_i$ et (7) est donc vérifiée ;
   si $\hat{x}_j = 0$ alors $\hat{z}_{ij} = 0$ et (7) est donc vérifiée.

ii) Montrons que $\sum_{i \in M} \hat{r}_i > \sum_{i \in M} r_i^*$
   (7) et (10) $\Longrightarrow$ $z_{ij}^* \geq r_i^* x_j^*$ $(i \in M,\, j \in N)$ donc, puisque $b_{ij} > 0$, (6') $\Longrightarrow$ $a_i + \sum_{j \in N} a_{ij} x_j^* \geq b_i r_i^* + \sum_{j \in N} b_{ij} r_i^* x_j^*$ $(i \in M)$ $\Longrightarrow$ $r_i^* \leq \frac{a_i + \sum_{j \in N} a_{ij} x_j^*}{b_i + \sum_{j \in N} b_{ij} x_j^*} \Longrightarrow \sum_{i \in M} r_i^* < \sum_{i \in M} \hat{r}_i$ puisque $f(\hat{x}) > f(x^*)$.

D'après (i) et (ii), si (x^*, r^*, z^*) est une solution optimale de $LIN2$ alors x^* est une solution optimale de SRH. □

3. Amélioration des reformulations linéaires

3.1. Amélioration de $LIN1$ ($LIN1_a$)

Pour améliorer la linéarisation $LIN1$, nous considérons à la fois une borne supérieure $\overline{y}_i$ et une borne inférieure $\underline{y}_i \geq 0$ de y_i. Ainsi, au lieu de remplacer le produit $y_i x_j$ par z_{ij}, on remplacera le produit $(y_i - \underline{y}_i)x_j$ par z_{ij}. De cette façon, nous dégageons une partie linéaire $(\underline{y}_i x_j)$ dans la fonction objectif et dans les containtes. Nous obtenons :

$$(LIN1_a) \begin{cases} \text{Max} \sum_{i=1}^{m} \left(a_i y_i + \sum_{j=1}^{n} a_{ij} z_{ij} + \sum_{j=1}^{n} a_{ij} \underline{y}_i x_j \right) & \\ b_i y_i + \sum_{j=1}^{n} b_{ij} z_{ij} + \sum_{j=1}^{n} b_{ij} \underline{y}_i x_j = 1 & i \in M \\ z_{ij} \geq y_i - \bar{y}_i(1 - x_j) - \underline{y}_i x_j & i \in M,\ j \in N \\ z_{ij} \leq y_i - \underline{y}_i & i \in M,\ j \in N \\ z_{ij} \leq (\bar{y}_i - \underline{y}_i)x_j & i \in M,\ j \in N \\ x_j \in \{0, 1\},\ z_{ij} \geq 0 & i \in M,\ j \in N. \end{cases}$$

La formulation $LIN1_a$ comprend le même nombre de variables et de contraintes que $LIN1$. Cependant, elle nécessite un pré-traitement supplémentaire, le calcul des coefficients $\underline{y}_i$ pour tout $i \in M$.

Proposition 3.1. *La valeur optimale de $\overline{LIN1}_a$, la relaxation continue de $LIN1_a$, est inférieure ou égale à la valeur optimale de $\overline{LIN1}$, la relaxation continue de $LIN1$.*

Démonstration. Soit $(\tilde{x}, \tilde{y}, \tilde{z})$ une solution optimale de $\overline{LIN1}_a$. On construit une solution $(\hat{x}, \hat{y}, \hat{z})$ admissible de $\overline{LIN1}$ donnant la même valeur à la fonction objectif en posant $\hat{z}_{ij} = \tilde{z}_{ij} + \underline{y}_i \tilde{x}_j$ $(i \in M,\ j \in N)$. □

3.2. Amélioration de $LIN2$ ($LIN2_a$)

L'idée de cette amélioration est similaire à celle que nous avons utilisée pour construire $LIN1_a$. Nous exploitons la connaissance d'une borne inférieure $\underline{r}_i \geq 0$ de r_i dans toute solution optimale de SRH. Ainsi, en remplaçant dans $LIN2$ le

produit $(r_i - \underline{r}_i)x_j$ par z_{ij} pour tout $i \in M$ et pour tout $j \in N$, on obtient :

$$(LIN2_a)\begin{cases} \text{Max} \sum_{i=1}^{m} r_i & \\ a_i + \sum_{j=1}^{n} a_{ij}x_j = b_i r_i + \sum_{j=1}^{n} b_{ij} z_{ij} + \sum_{j=1}^{n} b_{ij}\underline{r}_i x_j & i \in M \\ z_{ij} \geq r_i - \overline{r}_i(1 - x_j) - \underline{r}_i x_j & i \in M,\ j \in N \\ x_j \in \{0,\ 1\},\ z_{ij} \geq 0 & i \in M,\ j \in N. \end{cases}$$

$LIN2_a$ comprend le même nombre de variables et de contraintes que $LIN2$ mais nécessite le calcul de $\underline{r}_i$ pour tout $i \in M$.

Proposition 3.2. *La valeur optimale de $\overline{LIN2}_a$, la relaxation continue de $LIN2_a$, est inférieure ou égale à la valeur optimale de $\overline{LIN2}$, la relaxation continue de $LIN2$.*

Démonstration. La preuve est similaire à la preuve de la proposition 3.1. □

4. Élaboration des bornes supérieures et inférieures

Les différents modèles proposés nécessitent la connaissance de bornes inférieures et supérieures de certaines expressions dans toute solution optimale de SRH. Pour $LIN1$ et $LIN1_a$, on dispose de bornes évidentes pour y_i : $\overline{y}_i = 1/b_i$ et $\underline{y}_i = 1/(b_i + \sum_{j=1}^{n} b_{ij})$.

Pour calculer $\overline{r}_i$ et $\underline{r}_i$, les bornes de r_i utilisées dans $LIN2$ et $LIN2_a$, nous avons résolu pour tout $i \in M$, les 2 problèmes hyperboliques en 0-1 sans contrainte : $\text{Max/Min}\left\{\frac{a_i + \sum_{j=1}^{n} a_{ij}x_j}{b_i + \sum_{j=1}^{n} b_{ij}x_j} : x \in \{0,\ 1\}^n\right\}$. Il existe plusieurs méthodes permettant d'optimiser le ratio de deux fonctions linéaires. Nous avons choisi d'utiliser une approche par paramétrisation, l'algorithme de Isbell et Marlow [10]. Il s'agit d'un algorithme itératif très simple qui demande, à chaque itération, l'optimisation d'une fonction linéaire sans contrainte. Notons qu'il existe un algorithme de complexité linéaire pour résoudre ce problème [8]. Cependant, dans la résolution de SRH, le temps de pré-traitement consistant à optimiser les m ratios, un par un, est négligeable.

5. Expérimentations

Tous les programmes associés aux différentes linéarisations présentées ont été résolus en utilisant CPLEX 6.0 [9] sur un ordinateur de type Pentium II à 300 MHz. Les expérimentations ont été réalisées sur des problèmes engendrés aléatoirement. Les coefficients b_i, a_{ij} *et* b_{ij} sont des entiers uniformément distribués entre 10 et 20 et les coefficients a_i sont des entiers uniformément distribués entre 1 et 10.

TABLEAU 1. Comparaison expérimentale des 4 relaxations linéaires du problème SRH.

Prob		$\overline{LIN1}$		$\overline{LIN2}$		$\overline{LIN1_a}$		$\overline{LIN2_a}$	
m	***n***	***Val***	***Tps***	***Val***	***Tps***	***Val***	***Tps***	***Val***	***Tps***
10	30	13,2	0,3	12,8	0,2	**12,3**	**0,3**	12,8	0,3
10	30	12,7	0,3	12,5	0,2	**11,9**	**0,3**	12,3	0,2
10	30	12,8	0,2	12,4	0,2	**11,9**	**0,3**	12,3	0,2
20	50	27,5	3,4	26,8	2,2	**25,6**	**3,9**	26,7	1,9
20	50	27,9	3,7	27,3	1,8	**26,1**	**3,5**	27,2	1,7
20	50	27,3	2,5	26,7	1,9	**25,6**	**3,4**	26,6	2,7
30	50	41,8	12,5	40,9	3,9	**39,0**	**6,9**	40,8	4,5
30	50	40,6	7,3	39,6	4,0	**37,2**	**9,4**	39,5	4,6
30	50	41,0	4,5	40,0	3,8	**37,9**	**8,2**	39,9	4,4

Le tableau 1 compare les valeurs optimales des relaxations continues des programmes $LIN1$, $LIN2$, $LIN1_a$ et $LIN2_a$. Pour chaque relaxation nous donnons la valeur optimale (val) et le temps de résolution en secondes (Tps).

D'après le tableau 1, il est clair que la relaxation du programme $LIN1_a$ est la plus précise et que la relaxation de $LIN1$ est la moins précise au moins dans ces conditions expérimentales. Néanmoins, $LIN1_a$ est-elle la linéarisation la plus efficace et $LIN1$ la moins efficace pour résoudre le problème ?

Nous présentons dans le tableau 2 les résultats de la résolution exacte pour les quatre linéarisations envisagées. Chaque ligne du tableau représente la moyenne des résultats obtenus sur 5 instances du problème. Lorsque les 5 instances n'ont pas pu être toutes résolues dans la limite de temps, fixée à 10 000 secondes par instance, la moyenne porte sur les seules instances résolues. Pour chaque type d'instances, nous indiquons la moyenne de l'écart relatif entre la valeur de la relaxation continue du problème et la valeur de la solution optimale (GAP), la durée moyenne totale de résolution en secondes (Tps) et le nombre d'instances résolues (#).

Le tableau 2 montre que la linéarisation $LIN1_a$ est la plus efficace, ce qui confirme les résultats obtenus pour la relaxation continue. En revanche, ce sont les linéarisations $LIN2$ et $LIN2_a$ qui sont les moins efficaces et non pas $LIN1$. Pour $LIN2$ et $LIN2_a$, nous avons constaté que la convergence vers la solution optimale était très lente. Il est clair aussi que le temps de résolution croît considérablement avec la taille des instances. Cela peut s'expliquer, en partie, par le fait que plus l'instance est grande, plus l'écart relatif entre la solution exacte et la borne supérieure donnée par la relaxation continue est important. Le tableau 2 confirme également que l'écart relatif est moins important pour la linéarisation $LIN1_a$.

Nous proposons ci-après une approche heuristique permettant de résoudre rapidement, de manière approchée, de grandes instances de SRH. Cette solution heuristique est également exploitée pour accélérer la résolution exacte.

TABLEAU 2. Comparaison expérimentale de la résolution exacte du problème SRH par les 4 reformulations linéaires en variables mixtes.

Prob		$LIN1$			$LIN2$			$LIN1_a$			$LIN2_a$		
m	***n***	***GAP***	***Tps***	**#**	***GAP***	***Tps***	**#**	***GAP***	***Tps***	**#**	***GAP***	***Tps***	**#**
10	10	16	1,0	5	9	0,7	5	7	**0,5**	**5**	8,7	0,74	5
10	20	18,4	101,1	5	12,7	687,4	5	9,4	**39,1**	**5**	12,6	400,1	5
10	30	27,2	4322,6	1	23	-	0	18,9	**5223,3**	**3**	22,6	-	0
20	10	19	2,8	5	10,7	2,5	5	8,2	**1,6**	**5**	10,3	2,26	5
20	20	24	1343,4	5	19	-	0	14,5	**774,7**	**5**	18,5	2463	2
30	10	18	5,1	5	9,8	4,4	5	7,3	**3,6**	**5**	9,4	4,04	5
30	20	25	3755,1	4	19,3	-	0	14,3	**3232,5**	**5**	18,8	-	0

6. UNE MÉTHODE HEURISTIQUE

6.1. DESCRIPTION DE LA MÉTHODE

Il s'agit d'un algorithme de recherche locale. Partant de la solution admissible $x = (0, ..., 0)$, on essaye à chaque itération d'améliorer le plus possible la solution en modifiant la valeur d'une seule variable. On s'arrête lorsqu'on ne peut plus améliorer la solution de cette façon. Pour tout $x \in \{0,1\}^n$, notons x^j le vecteur de $\{0,1\}^n$ obtenu à partir de x en modifiant uniquement la j-ème composante. Ainsi si $\hat{x} = (\hat{x}_1, ..., \hat{x}_j, ..., \hat{x}_n)$ alors $\hat{x}^j = (\hat{x}_1, ..., 1 - \hat{x}_j, ..., \hat{x}_n)$. Rappelons que $f(x) = \sum_{i=1}^{m} \frac{a_i + \sum_{j=1}^{n} a_{ij} x_j}{b_i + \sum_{j=1}^{n} b_{ij} x_j}$.

Heuristique

(1) $x = (0, 0, ..., 0)$;
(2) déterminer l'indice $s \in N$ tel que $f(x^s) = \max\{f(x^j) : j \in N\}$;
(3) si $f(x^s) > f(x)$ alors $x \leftarrow x^s$; retour en 2.
Sinon STOP (x est la solution heuristique)
finsi

Les expérimentations ont montré que la solution admissible obtenue par cette heuristique était de très bonne qualité (égale à la solution optimale dans la plupart des cas). Elle est également utile pour la résolution exacte. En fournissant la borne inférieure correspondant à cette solution admissible au logiciel de programmation linéaire en variables mixtes, nous obtenons une réduction de l'arbre d'exploration avec une diminution du temps de calcul. Cette valeur de la solution heuristique est également utilisée pour obtenir des bornes plus précises pour les variables y_i et r_i intervenant dans les différentes formulations linéaires (voir paragraphe 7).

TABLEAU 3. Comparaison expérimentale entre $LIN1_a$, l'heuristique proposée et $LIN1_a$ avec la valeur de la solution heuristique.

		cplex sans sol_initiale				**cplex avec sol_initiale**		
Prob		$LIN1_a$			**Heuristique**	$LIN1_a$		
m	***n***	***Nb_nd***	***Tps(s)***	**#**	***Tps***	***Nb_nd***	***Tps(s)***	**#**
5	10	41,2	0,1	5	**0,0003**	31,8	0,1	5
5	20	1087,8	3,9	5	**0,0005**	920,6	3,1	5
5	30	67177	324,5	5	**0,0009**	56631	236,2	5
10	10	141,2	0,6	5	**0,0009**	131,2	0,6	5
10	20	20574	147,2	5	**0,0019**	18302	127	5
10	30	340451	4379,1	3	**0,0016**	400511	4374	4
20	10	95,2	1,4	5	**0,0017**	91	1,4	5
20	20	37281	880,6	5	**0,0048**	33851	762,2	5

6.2. Résultats expérimentaux

Afin de déterminer une solution approchée, nous avons implanté en langage C sur un Pentium II à 300 MHz l'heuristique très simple présentée ci-dessus. Les tests ont été réalisés sur des problèmes engendrés aléatoirement comme dans le paragraphe 5. Ces tests concernent uniquement l'heuristique proposée et son influence sur la résolution du problème SRH par $LIN1_a$. Le tableau 3 présente les résultats correspondants. Chaque ligne du tableau représente la moyenne des résultats obtenus sur 5 instances du problème. Pour $LIN1_a$ avec et sans l'utilisation de la valeur de la solution heuristique, nous précisons le nombre de nœuds (Nb_nd), le temps CPU (Tps) et le nombre d'instances que nous avons pu résoudre (#). Pour toutes les instances où la solution optimale est connue (obtenue par une des linéarisations), l'heuristique permet de déterminer cette solution optimale. Nous indiquons également dans le tableau 3 le temps de calcul (TPs) requis par l'heuristique. On constate qu'il est excessivement court. On constate également que la connaissance d'une bonne solution améliore légèrement la résolution du problème à travers la formulation $LIN1_a$.

Nous proposons dans le paragraphe suivant une méthode de raffinement des bornes des variables y_i et r_i permettant d'accélérer la résolution exacte. Cette méthode exploite la connaissance d'une solution heuristique.

7. Raffinement des coefficients des modélisations et résultats expérimentaux

Nous montrons dans ce paragraphe comment obtenir des bornes inférieures et supérieures plus précises pour les variables y_i et r_i, $i \in M$. Le paragraphe 7.1

décrit la démarche adoptée. Les résultats expérimentaux prouvant l'intérêt de la méthode sont présentés au paragraphe 7.2.

7.1. Description de la méthode

Nous avons proposé au paragraphe 4 des valeurs pour $\underline{y}_i$, $\overline{y}_i$, $\underline{r}_i$ et $\overline{r}_i$. Nous proposons ici des valeurs plus précises. Pour les calculer nous introduisons les 2 contraintes $\sum_{j=1}^{n} x_j \geq K_1$ et $\sum_{j=1}^{n} x_j \leq K_2$ où K_1 et K_2 correspondent respectivement au nombre minimal et maximal de variables x_j non nulles à l'optimum du problème SRH.
Pour $k = 0, ..., n$, posons $B_k = \sum_{i=1}^{m} \left(\text{Max} \left\{ \frac{a_i + \sum_{j=1}^{n} a_{ij} x_j}{b_i + \sum_{j=1}^{n} b_{ij} x_j} : x \in \{0,1\}^n, \sum_{j=1}^{n} x_j = k \right\} \right)$.
B_k est une borne supérieure de SRH sous la contrainte $\sum_{j=1}^{n} x_j = k$.

Pour déterminer K_1, on calcule B_k pour $k = 0$, 1, etc. et on s'arrête dès que $B_k \geq \alpha$ où α designe la valeur de la solution heuristique. En d'autres termes $K_1 = \min\{k : k \in \{0, 1, ..., n\}, B_k \geq \alpha\}$. De la même façon $K_2 = \max\{k : k \in \{0, 1, ..., n\}, B_k \geq \alpha\}$. Posons $X = \{x : x \in \{0,1\}^n, \sum_{i=1}^{n} x_i \geq K_1, \sum_{i=1}^{n} x_i \leq K_2\}$. $\underline{y}_i$ et $\overline{y}_i$ sont alors données par la résolution des 2 programmes hyperboliques sous contraintes Min/Max $\left\{ 1 \Big/ \left(b_i + \sum_{j=1}^{n} b_{ij} x_j \right) : x \in X \right\}$, et $\underline{r}_i$ et $\overline{r}_i$, par la résolution des 2 programmes Min/Max $\left\{ \left(a_i + \sum_{j=1}^{n} a_{ij} x_j \right) \Big/ \left(b_i + \sum_{j=1}^{n} b_{ij} x_j \right) : x \in X \right\}$.
Pour résoudre ces programmes, nous avons également utilisé la méthode par paramétrisation, qui est très efficace, en résolvant les programmes linéaires en 0-1 associés avec CPLEX.

7.2. Résultats expérimentaux

Nous présentons dans le tableau 4 des résultats expérimentataux permettant de comparer la résolution de SRH par $LIN1_a$ couplée avec l'heuristique, avec et sans l'étape de raffinement. Les expérimentations ont été effectuées avec les mêmes instances que celles qui sont décrites au paragraphe 5 et également avec des instances de plus grande taille engendrées de la même façon.

Chaque ligne du tableau représente la moyenne des résultats concernant 5 instances. Pour chaque type de résolution, nous indiquons le nombre de nœuds développés dans le branch-and-bound (Nb_nd), la durée de la résolution, étape de raffinement incluse (TPS) ainsi que le nombre d'instances résolues parmi les 5 considérées (#). Le temps de résolution d'une instance est limité à 10 000 s.

Il est clair d'après le tableau 4 que l'étape de raffinement est très efficace. En effet, en améliorant les bornes des variables y_i dans $LIN1_a$, nous obtenons une réduction de l'arbre d'exploration et une accélération de la résolution. Par exemple, pour les instances (10×30), $LIN1_a$ sans raffinement permet de résoudre 4 instances en un temps moyen de 4400 s., alors que $LIN1_a$ avec raffinement permet de résoudre 4 instances en moins de 2500 secondes en moyenne. On note également que plus le nombre de variables est grand, plus l'étape de raffinement est efficace (voir les instances 4×50 et 2×100).

TABLEAU 4. Comparaison expérimentale de la formulation $LIN1_a$ avec et sans raffinement des coefficients du modèle.

Problème		$LIN1_a$ **sans raffinement**			$LIN1_a$ **avec raffinement**		
m	***n***	***Nb_nd***	***Tps(s)***	#	***Nb_nd***	***Tps(s)***	#
5	10	31,8	0,1	5	**23,4**	**0,1**	**5**
5	20	920,6	3,1	5	**490,4**	**1,9**	**5**
5	30	56631	236,2	5	**31823**	**137,2**	**5**
10	10	131,2	0,6	5	**111,3**	**0,5**	**5**
10	20	18302	127	5	**14490,8**	**98,6**	**5**
10	30	400511	4374	4	**210677,5**	**2463,9**	**4**
20	10	91	1,4	5	**57,25**	**1,1**	**5**
20	20	33851	762,2	5	**25626**	**697,5**	**5**
4	50	216832	1171,5	5	**73694,7**	**401**	**5**
2	100	288901,3	1252,9	4	**181902**	**404,8**	**5**

8. UN ALGORITHME DE BRANCH-AND-BOUND AVEC RÉAJUSTEMENT PROGRESSIF DES COEFFICIENTS DES CONTRAINTES

Nous avons évoqué au paragraphe précédent l'importance du choix des coefficients $\underline{y}_i$, $\overline{y}_i$, $\underline{r}_i$ et $\overline{r}_i$ dans les reformulations linéaires proposées vis-à-vis de l'efficacité de la résolution de SRH par un logiciel standard de PLVM. C'est pourquoi nous proposons maintenant un algorithme de branch-and-bound avec réajustement progressif de ces coefficients en chaque nœud de l'arbre de recherche. En effet, les PLVM que nous avons à résoudre sont obtenus par linéarisation de programmes mathématiques non linéaires. Ces linéarisations engendrent des coefficients dont les valeurs ne sont pas définies de façon unique. Elles doivent seulement vérifier certaines conditions. Ce sont ces coefficients qu'il faut choisir le mieux possible (de façon à obtenir une bonne relaxation continue). Ce sont également eux que l'on peut théoriquement réajuster en chaque chaque nœud de l'arbre de recherche. Ces réajustements des coefficients, comme leur calcul à la racine de l'arbre, peuvent être plus ou moins complexes.

8.1. DESCRIPTION DE L'ALGORITHME

L'idée de base de cet algorithme consiste à réajuster certains coefficients du modèle en chaque nœud de l'arbre de recherche. Dans une procédure de branch-and-bound standard, l'unique différence entre le sous-problème à résoudre au nœud père et celui à résoudre au nœud fils est le remplacement de la contrainte $(0 \leq x_s \leq 1)$ par la contrainte de fixation ($x_s = 1$ ou $x_s = 0$) pour un certain indice $s \in N$. Dans notre procédure spécifique, on va pouvoir faire évoluer certains

coefficients du modèle en tenant compte des variables déjà fixées. La réécriture du sous-problème fils se fait donc non seulement par l'ajout d'une contrainte de fixation d'une variable mais aussi par la modification de certains coefficients.

La seule différence entre notre algorithme et une procédure de branch-and-bound standard consiste donc à recalculer les coefficients $\underline{y}_i$, $\overline{y}_i$, $\underline{r}_i$ et $\overline{r}_i$ en chaque nœud de l'arbre de recherche. Ces calculs se font par la même procédure que celle que nous avons décrite au paragraphe 4 mais en tenant compte des variables déjà fixées dans la branche correspondante. En ce qui concerne la variable de séparation, nous avons choisi, après plusieurs essais, de prendre la variable dont la valeur est la plus petite .

Après de nombreuses expériences numériques, nous avons constaté que notre procédure arborescente spécifique développait beaucoup moins de nœuds que le solveur standard. En revanche, l'amélioration en temps de résolution était faible. En effet, le temps nécessaire à l'évaluation en chaque nœud est plus important dans notre méthode. Contrairement à un algorithme de branch-and-bound standard, pour résoudre un problème d'un nœud fils, nous ne pouvons pas exploiter la base réalisable du problème résolu à son nœud père. Pour accélérer notre procédure, nous avons choisi de limiter le nombre de réajustements des coefficients. Cela entraîne une augmentation du nombre de nœuds développés mais procure un gain important en temps de résolution puisque, pour certains nœuds, nous utilisons les bases réalisables de leur nœud père. De plus, il est apparu qu'il n'était pas intéressant de réajuster les coefficients si le nombre de variables fixées était encore petit, les nouveaux coefficients étant généralement peu différents des précedents.

8.2. Résultats numériques et comparaisons

Nous indiquons dans le tableau 5 les résultats concernant 4 approches pour la résolution du problème SRH : $LIN1_a$ et $LIN2_a$ avec un branch-and-bound standard (cplex) couplé avec l'heuristique et l'étape de raffinement, puis $LIN1_a$ et $LIN2_a$ avec le branch-and-bound spécifique décrit ci-dessus. Ces résolutions ont été implémentées en langage C sur un ordinateur Pentium II à 300 MHz. Les expérimentations ont été réalisées dans les mêmes conditions que celles qui sont décrites dans le paragraphe 5. Chaque ligne correspond à une instance et pour chaque type d'instances, nous indiquons le nombre total de nœuds et le temps CPU.

Le tableau 5 montre que la résolution de $LIN1_a$ et $LIN2_a$ avec la procédure spécifique proposée est très efficace par rapport à la procédure standard. On note également que, contrairement à la résolution standard où le programme $LIN1_a$ donne toujours les meilleurs résultats, c'est $LIN2_a$ qui fournit les meilleurs résultats dans le cas du branch-and-bound spécifique. En utilisant le branch-and-bound standard, $LIN2_a$ converge très lentement vers la solution optimale. Par contre, avec notre procédure spécifique et grâce au réajustement progressif des coefficients, la convergence devient nettement plus rapide et meilleure que celle de $LIN1_a$. Nous pouvons donc conclure intuitivement que $LIN2_a$ est beaucoup plus sensible aux valeurs des bornes que $LIN1_a$.

TABLEAU 5. Comparaison expérimentale entre le branch-and-bound standard de cplex et notre branch-and-bound spécifique.

		Cplex+Sol_ initiale+Raf				**Branch&Bound Spécifique**			
Prob		$LIN1_a$		$LIN2_a$		$LIN1_a$		$LIN2_a$	
m	**n**	**Nb**	**Tps**	**Nb**	**Tps**	**Nb**	**Tps**	**Nb**	**Tps**
10	20	5736	46	35142	170	2089	64	**2311**	**29**
10	20	5797	49	29806	153	1843	57	**2709**	**24**
10	20	2106	18	27296	166	1421	38	**1179**	**14**
20	20	24894	705	81460	1088	4045	497	**5337**	**185**
20	20	24715	616	76610	1021	3199	399	**4161**	**142**
20	20	40956	1034	99610	1440	4627	522	**6917**	**228**
30	20	35315	1869	95160	3628	4143	961	**5267**	**356**
30	20	17559	1019	79321	2063	3609	836	**4613**	**310**
30	20	15058	972	26838	876	3001	748	**3183**	**270**

9. CONCLUSION

Nous avons consideré dans cet article le problème de la maximisation de la somme de plusieurs ratios hyperboliques en 0-1. Il s'agit d'un problème d'optimisation combinatoire fractionnaire relativement peu étudié jusqu'à présent. Nous avons proposé différentes reformulations linéaires de ce problème qui se distinguent par les expressions linéarisées et les contraintes ajoutées. Ces reformulations sont très faciles à exploiter lorsque l'on dispose d'un outil de programmation linéaire en variables mixtes. De plus, elles pourraient être appliquées au cas du problème SRH sous contraintes linéaires. De nombreux autres problèmes classiques de programmation quadratique en variables mixtes pourraient aussi être abordés dans ce même esprit. Nous avons ensuite proposé deux stratégies de résolution. La première consiste à résoudre directement le programme en utilisant un outil général de programmation linéaire en variables mixtes. La deuxième consiste à résoudre le problème par un algorithme de branch-and-bound spécifique fondé sur une reformulation plus précise en chaque nœud de l'arbre de recherche. En d'autres termes, on peut dire qu'en chaque nœud de l'arbre de recherche on essaye d'améliorer la linéarisation du problème associé à ce nœud ; on ne se contente pas de la linéarisation initiale (à la racine de l'arbre). Nous avons également proposé pour la résolution de SRH une méthode heuristique, simple mais très efficace en pratique. Les linéarisations proposées et les différentes stratégies envisagées pour les résoudre ont été comparées expérimentalement. Ces comparaisons ont montré que la résolution du problème par le branch-and-bound spécifique appliqué à l'une des deux linéarisations et couplé avec la méthode heuristique était relativement efficace. Cette approche permet de traiter facilement des instances comportant une trentaine de ratios et une vingtaine de variables. Ces résultats fournissent un

point de départ intéressant pour tester l'efficacité de nouveaux algorithmes puisqu'à notre connaissance aucun résultat expérimental n'a été publié sur ce problème fractionnaire.

Références

[1] S.R. Arora, K. Swarup and M.C. Puri, The set covering problem with linear fractional functional. *Indian J. Pure Appl. Math.* **8** (1977) 578–588.

[2] A. Billionnet, Approximate and exact solution methods for the hyperbolic 0-1 knapsack problem. *Inform. Syst. Oper. Res.* **40** (2002) 97–110.

[3] K. Djebali, *Modélisation et résolution de problèmes d'optimisation combinatoire par la programmation linéaire en variables mixtes.* rapport de Thèse, CNAM, Paris (2003).

[4] J.E. Falk and S.W. Paloscay, Optimising the sum of linear fractional functions. *Adv. Global Optim.* (1992) 221–258.

[5] R.W. Freund and F. Jarre, Solving the sum-of-ratios problem by an interior-point method. *J. Global Optim.* **19** (2001) 83–102.

[6] P.C. Gilmore and R.E. Gomory, A linear programming approach to the cutting stock problem. *Oper. Res.* **11** (1963) 52–53.

[7] P. Hansen, M.V. Poggi De Aragao and C.C. Ribeiro, Boolean query optimization and the 0-1 hyperbolic sum problem. *Ann. Math. Artif. Intell.* **1** (1990) 97–109.

[8] P. Hansen, M.V. Poggi De Aragao and C.C. Ribeiro, Hyperbolic 0-1 programming and information retrieval. *Math. Program.* **52** (1991) 255–263.

[9] Ilog, Inc., Cplex Division, *Using the cplex callable Library.* Version 6.0 (1998).

[10] J.R. Isbell and W.H. Marlow, Attribution games. *Naval Res. Logistics Quart.* **3** (1956) 71–94.

[11] H. Li, A Global approach for general fractional programming. *Eur. J. Oper. Res.* **73** (1994) 590–596.

[12] A. Nagih and G. Plateau, A partition algoritm for 0-1 hyperbolic programming problems. *Investigacion Operativa* **9** (1999) 167–178.

[13] A. Nagih and G. Plateau, Problèmes fractionnaires : applications et méthodes de résolution. *RAIRO Oper. Res.* **33** (1999) 383-419.

[14] T. Radzik, Fractional combinatorial optimization, in *Handb. Combin. Optim.*, edited by Z.-Z. Du and P. Paradaloas Kluwer Academic Publishers (1998) 429–478.

[15] A.L. Saipe, Solving a 0-1 hyperbolic program by branch-and-bound. *Naval Res. Logist. Quart.* **22** (1975) 397–416.

[16] S. Schaible and T. Ibaraki, Fractional programming. *Eur. J. Oper. Res.* **12** (1983) 325–338.

[17] S. Schaible, Fractional programming, in *Handb global Optim.*, edited by R. Horst and P. Paradalos. Kluwer Academic Publishers (1995) 495–608.

[18] L. Scharge, LINDO USER'S MANUAL : Release 5.0, The Scientific Press, San Francisco (1991).

[19] I.M. Stancu-Minasian, *Fractional Programming.* Kluwer Academic Publishers (1997).

[20] T. Wu, A note on a global approach for general 0-1 fractional programming. *Eur. J. Oper. Res.* **101** (1997) 229–223.