CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS

Thèse présentée pour l'obtention du titre de

DOCTEUR DU CNAM

Spécialité : **INFORMATIQUE**

par

Karima DJEBALI

Modélisation et Résolution de problèmes d'optimisation combinatoire par la programmation mathématique en variables mixtes

soutenue le 19 décembre 2003 devant le jury composé de :

Directeur de thèse :	**M. Alain BILLIONNET** (I.I.E. - CNAM)
Rapporteurs :	**M. Abdel LISSER** (Université Paris 11)
	M. Gerard PLATEAU (Université Paris 13)
Examinateurs :	**M. Van-Dat Cung** (ENSGI-INPG de Grénoble)
	Mme Sourour Elloumi (CNAM)
	M. Arnaud Fréville (Université de Valenciennes)

Centre d'Études et De Recherche en Informatique du Cnam
(CEDRIC)

Modélisation et résolution de problèmes d'optimisation combinatoire par la programmation mathématique en variables mixtes

Table des matières

Introduction générale

De nombreux problèmes de la recherche opérationnelle peuvent être décrits par un modèle mathématique, c'est à dire un modèle qui utilise les mathématiques pour décrire ce que l'on cherche à optimiser ainsi que les relations entre les différentes données du problème. Les problèmes considérés seront généralement des problèmes de décision, c'est à dire des problèmes comportant des variables correspondant à des décisions. Ces décisions sont prises pour atteindre un objectif particulier et le modèle comportera une mesure explicite de l'atteinte de cet objectif. Cette mesure est appelée fonction objectif. Une fois que ce modèle mathématique est construit, on essaye de formaliser le problème par un modèle mathématique plus précis, un programme mathématique. De façon générale, nous dirons qu'un programme mathématique définit une fonction f à optimiser, les variables de cette fonction étant soumises à des contraintes pouvant s'exprimer sous la forme $g_i \leq 0$, les fonctions f et g_i étant des fonctions quelconques. Certaines contraintes sur les variables seront souvent décrites séparément : certaines variables doivent prendre des valeurs réelles positives ou nulles, d'autres doivent prendre des valeurs entières, d'autres encore ne peuvent prendre que les valeurs 0 ou 1.

Le travail est déjà bien avancé lorsqu'on a réussi à décrire le problème d'optimisation considéré par un programme mathématique, mais le problème n'est pas résolu pour autant. Sa résolution effective peut être très difficile. Cette difficulté peut provenir de plusieurs aspects du programme. Le nombre de variables d'un programme mathématique décrivant certains problèmes d'optimisation concrets peut facilement atteindre plusieurs milliers. Par ailleurs, le fait que certaines variables ne puissent prendre que des valeurs entières est également une grande source de difficultés. Malheureusement, en recherche opérationnelle, les variables de décision sont souvent des variables entières. Un troisième type de difficulté provient de la nature des fonctions intervenant dans la fonction économique et dans l'expression des contraintes. Ces fonctions peuvent être non linéaires, non convexes et même ne pas admettre de formes algébriques explicites. Certains programmes mathématiques peuvent être si compliqués qu'il est pratiquement impossible de les résoudre malgré la puissance des ordinateurs disponibles sur le marché.

Nous nous intéressons principalement dans ce travail aux problèmes d'optimisation pouvant s'énoncer, plus ou moins facilement, sous la forme de programmes mathématiques particuliers où la fonction économique ainsi que les fonctions intervenant dans les contraintes sont des fonctions linéaires. Dans le cas général, nous appellerons ces programmes mathématiques ***programmes linéaires en variables mixtes*** et nous les noterons en abrégé **PLVM**. Cette appellation traduit d'une part la linéarité du problème et, d'autre part, le fait que les variables soient continues ou discrètes. Lorsqu'on étudie un problème d'optimisation deux cas seront envisageables : la formulation naturelle du problème est déjà un PLVM ou, au contraire, sa formulation naturelle n'est pas linéaire. Nous nous intéresserons plus particulièrement au second cas et proposerons, pour certains programmes mathématiques non linéaires, une ou plusieurs transformations en un PLVM. Essayer de résoudre un programme mathématique non linéaire en le transformant en un programme linéaire en variables mixtes n'est pas toujours une bonne approche mais nous verrons que, dans de nombreux cas, l'approche peut être très intéressante. Une des motivations importantes pour se ramener à un PLVM est qu'il existe aujourd'hui de nombreux logiciels commerciaux, appelés quelquefois *solveurs*, extrêmement performants pour résoudre ces programmes. Les problèmes d'optimisation linéaires en variables mixtes sont en effet des problèmes bien connus, étudiés depuis longtemps et dont la résolution a fait des progrès considérables durant les quinze dernières années. Ainsi grâce à ces logiciels et à la puissance des ordinateurs actuels, on peut résoudre des PLVM comportant des milliers de variables. Nous nous intéresserons également à plusieurs problèmes en variables mixtes pouvant s'énoncer comme la minimisation d'une fonction quadratique convexe sous des contraintes linéaires (PQVM). Il existe en effet des algorithmes très efficaces pour résoudre la relaxation continue de ces programmes et les solveurs les plus courants offrent cette fonctionnalité.

Une difficulté importante lorsqu'on essaye de formuler un problème d'optimisation par un PLVM ou un PQVM est qu'il existe généralement plusieurs formulations. Si ces formulations sont équivalentes dans le sens où elles expriment correctement le problème initial, elles ne le sont pas forcément en ce qui concerne la résolution pratique, numérique, du problème. On verra dans les différents chapitres de cette thèse des exemples de modélisations différentes (mais valides) d'un même problème qui engendrent des temps de résolution très différents.

Une fois que le problème d'optimisation considéré est formalisé par un PLVM (ou un PQVM), il reste encore une dernière étape à effectuer avant sa résolution proprement dite. Cette étape est souvent appelée pré-traitement. Ce pré-traitement consiste à affiner la formulation dans le but d'améliorer la résolution du problème, c'est à dire finalement, de diminuer le temps de calcul nécessaire à sa résolution. Dans certains cas, ce pré-traitement est crucial dans la mesure où, son absence peut rendre impossible la résolution du problème. Ces pré-traitements peuvent être plus ou moins complexes. Ils peuvent consister, par exemple, à effectuer différents raisonnements pour fixer définitivement dans le programme la valeur de certaines variables. Ils peuvent aussi consister à ajouter certaines contraintes, redondantes en ce qui concerne la formulation du problème initial, mais qui seront d'une grande utilité dans la résolution numérique du problème. Un autre pré-traitement possible est la détermination, par une méthode approchée, d'une bonne solution du problème initial. Nous proposons dans cette thèse un certain nombre de pré-traitements. Il est à noter que certains pré-traitements sont effectués de façon automatique par les solveurs commerciaux.

Au premier chapitre nous étudions le problème du plus court chemin robuste. C'est un problème important de la recherche opérationnelle qui a de nombreuses applications, par exemple dans le domaine des transports et des télécommunications. Il permet de formuler des problèmes de cheminement dans les réseaux en faisant intervenir une certaine incertitude sur les données (sur les valeurs attribuées aux arcs du graphe) sous la forme de scénarios. Cependant, contrairement au problème du plus court chemin classique, c'est un problème de type minmax NP-difficile. Nous passons en revue les différents travaux effectués dans le domaine, nous présentons de façon précise un algorithme pseudo-polynommial, de Yu et Yang, fondé sur la programmation dynamique. Puis nous proposons une approche complètement différente de ce problème en utilisant la programmation linéaire en variables mixtes (PLVM). Nous avons implémenté ces deux algorithmes afin de comparer précisément leurs performances. Les expérimentations menées sur des instances engendrées aléatoirement ont montré que le temps de calcul et l'espace mémoire requis par l'algorithme de Yu et Yang ne permettaient pas de traiter de grandes instances. Il est par exemple très difficile de traiter des graphes comportant quelques dizaines de sommets avec 4 scénarios. En revanche l'approche par la PLVM permet de traiter sans difficulté des instances comportant une centaine de sommets et une trentaine de scénarios. Nous avons également comparé l'efficacité de notre algorithme avec un algorithme branch and bound spécifique proposé dans la littérature, celui de Kouvelis et Yu. Ces auteurs ont expérimenté leur algorithme sur des graphes en niveaux. Nous avons donc effectué nos comparaisons sur le même type de graphes. Notre approche permet de traiter des problèmes de taille beaucoup plus grande. Mentionnons par exemple que, pour des graphes comportant 40 sommets et 10 scénarios, la PLVM nécessite en moyenne moins d'une seconde de calcul alors que l'algorithme de Kuvelis et Yu en demande, en moyenne, 270. La PLVM semble donc être une approche très efficace pour ces problèmes de cheminement optimal avec scénarios.

Nous étudions au chapitre 2 le problème bien connu de la gestion optimale d'un portefeuille. Après une présentation rapide du domaine, nous considérons plus particulièrement deux aspects importants de cette question : 1. Comment construire un portefeuille optimal en choisissant un nombre fixe de titres parmi un ensemble de titres disponibles. Plus précisément, on cherche à déterminer un ensemble de portefeuilles efficaces vis à vis des deux critères : espérance de rendement et risque, le risque étant mesuré par la variance du rendement ; 2. Comment construire un portefeuille optimal à partir d'un portefeuille déjà constitué. Les critères à prendre en compte dans ce cas sont, comme précédemment, l'espérance de rendement et le risque, mais s'ajoutent également les coûts de transactions liés aux achats et aux ventes de titres nécessaires pour atteindre le portefeuille cible. Ces coûts de transaction comportent une partie linéaire et une partie fixe. Nous formulons ces problèmes comme des programmes quadratiques en variables mixtes. Nous proposons une méthode branch and bound pour les résoudre de façon exacte. L'évaluation en chaque nœud de l'arbre de recherche est fondée sur la résolution de la relaxation continue du programme quadratique en variables mixtes qui est un problème quadratique convexe. Pour les instances de grande taille, nous proposons des heuristiques fondées sur la résolution de cette relaxation continue couplée avec une procédure branch and bound tronquée. Les résultats expérimentaux montrent l'efficacité de cette approche heuristique (qualité de la solution et temps de calcul), pour le traitement des instances construites sur les données des cinq marchés mondiaux, par rapport à d'autres méthodes présentées dans la littérature.

Le troisième chapitre traite un problème d'optimisation combinatoire fractionnaire : la maximisation de la somme de plusieurs ratios hyperboliques en variables bivalentes. Contrairement à la maximisation d'un seul ratio, très peu d'auteurs se sont intéressés à ce problème qui semble difficile et dépourvu, a priori, de "belles propriétés". Il a cependant des applications importantes dans plusieurs domaines tels que les bases de données, la programmation stochastique et l'économie. Notre recherche sur ce problème vise, d'une part, à trouver une bonne formulation pour le résoudre par la programmation linéaire en variables mixtes, et d'autre part, à construire une heuristique pour le résoudre de façon approchée, la résolution exacte n'étant plus envisageable pour des instances de grande taille. Nous proposons dans ce chapitre une heuristique pour traiter le problème ainsi que trois formulation par la PLVM. Pour chacune de ces formulations nous proposons trois stratégies de résolution : une résolution directe utilisant le logiciel CPLEX, un outil général de PLVM, une résolution directe par CPLEX mais qui prend compte, cette fois, d'une bonne solution fournie par la méthode heuristique et enfin une résolution par un algorithme branch-and-bound spécifique fondé sur une réécriture plus précise des programmes en chaque nœud de l'arbre de recherche ; cette réécriture consistant à ajuster progressivement les coefficients des programmes. Toutes ces approches et stratégies ont été comparées expérimentalement. Ces comparaisons ont montré que la résolution du problème par le branch and bound spécifique appliqué à l'une des trois 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.

Enfin, dans le chapitre 4 nous considérons un problème très classique de l'optimisation combinatoire, le problème de l'ensemble stable de cardinal maximal d'un graphe. Nous passons tout d'abord en revue les principales méthodes qui ont été proposées pour résoudre ce problème. Cette présentation est loin d'être exhaustive car les publications sur le sujet sont extrêmement abondantes. Le but de notre recherche concernant ce problème est d'étudier ce que la programmation linéaire en variables binaires peut apporter à sa résolution. Pour cela, nous proposons différentes formulations de ce problème qui diffèrent par le nombre et la nature des contraintes impliquées. Nous présentons ensuite une étude expérimentale menée sur un grand nombre d'instances comportant 100 ou 200 sommets, et une étude comparative de ces résultats avec ceux de la littérature. La conclusion est que la programmation linéaire en variables binaires est une alternative intéressante pour résoudre ce problème. Elle permet en effet d'obtenir facilement des résultats comparables à des méthodes spécifiques très élaborées qui demandent une implémentation soigneuse pour être efficace. En revanche, par notre approche, il suffit d'utiliser un outil standard de programmation linéaire en nombres entiers.

Chapitre 1

Problème du plus court chemin robuste

1.1 Introduction

Les problèmes de cheminement constituent un thème classique en recherche opérationnelle dont les applications sont très nombreuses, notamment en transport et en télecommunication. Nous considérons, dans ce chapitre, l'un des plus importants problèmes de cheminement : le problème du plus court chemin. Soit un graphe $G = (V, A)$, où $V = \{v_1, ..., v_n\}$ est l'ensemble des nœuds et $A = \{a_1, ..., a_m\}$, l'ensemble des arcs. À chaque arc a_i de G est associé un coût c_{a_i}. Le problème du plus court chemin consiste à trouver un chemin entre un sommet source s donné et un sommet destination t donné qui minimise la somme des coûts des arcs. Il existe plusieurs algorithmes pour résoudre ce problème, le plus connu étant celui de Dijkstra [Dijkstra, 1959]. Cependant, son application est limitée au cas classique du problème, c'est-à-dire au cas où tous les coûts sont fixés et positifs. Dans la pratique, la recherche d'un chemin optimal dans un graphe est souvent sujette à certaines incertitudes et donc, les coûts des arcs ne sont pas toujours parfaitement déterminés. Par exemple, si on cherche à minimiser le temps de parcours entre deux villes, le facteur important n'est pas la distance mais le temps. Or le temps mis pour aller d'une ville à une autre dépend de multiples facteurs : heure, saison, accidents, état des routes, etc. Ainsi le temps de parcours peut varier de manière drastique.

Il arrive souvent, dans les modèles de recherche opérationnelle ou d'aide à la décision, que les données ou les paramètres nécessaires à l'utilisation d'un modèle ne sont pas déterminés de façon unique. Les coûts des ressources, les demandes de produits ou les revenus sont des exemples de données incertaines. Plusieurs jeux de valeurs (scénarios) sont plausibles. Malgré ces incertitudes, l'analyste devra fournir des solutions robustes et fiables, c'est-à-dire, de bonne qualité pour les différents jeux de valeurs. On trouve dans la littérature différentes façons pour traiter les incertitudes des données. On peut transformer le problème incertain en un problème risqué (l'incertitude en risque) en attribuant une probabilité subjective à chaque jeu de valeurs, et utiliser ensuite l'optimisation stochastique pour le résoudre. La difficulté d'estimer les probabilités à attribuer ne favorise pas cette approche. Il est aussi possible de transformer le problème incertain en un problème certain, c'est-à-dire déterministe, par exemple, en utilisant une estimation subjective des données. Cependant, la performance de la solution optimale obtenue est de très faible qualité.

De nombreux auteurs ont dèjà confirmé que ni l'optimisation stochastique ni l'optimisation déterministe ne sont adaptées aux problèmes incertains. Voici par exemple une citation de Yu et Yang [Yu et al., 1998] "Ample evidence exist in research litterature that for decision making environments in the prescence of significant data uncertainty in the input of the decision model, either the deterministic optimisation or the stochastic optimisation approach may not accurately represent the aim of the decision maker (see [Gupta et al., 1972], [Rosenhead et al., 1972], [Sengupta, 1991], [Kouvelis et al., 1992] and [Daniels et al., 95])".

Dans ce travail, pour résoudre le problème du plus court chemin avec plusieurs jeux de valeurs, nous considérons une approche robuste, une approche minimax. Nous adoptons une approche par scénarios pour caractériser l'incertitude des données. Á chaque scénario correspond une instance du problème. Dans la section 1.2, nous donnons la définition générale d'une approche robuste ainsi que différents critères de robustesse. Dans la section 1.3.4, nous présentons notre approche pour résoudre le problème. Enfin, des expérimentations numériques permettant de montrer sa performance sont données.

1.2 Approche robuste

Comme [Daniels et al., 95], [Deodhare et al., 1998], [Hermann, 1999], [Kouvelis et al., 1992], [Veeravalli et al., 1994] et [Yu et al., 1998], nous utilisons une approche robuste, plus précisément une approche minimax, pour résoudre un problème de décision dans un environnement incertain. L'objectif d'une telle approche est de trouver pour un ensemble de jeux de valeurs une solution raisonnable qui garantit une bonne performance pour n'importe quel jeu de valeurs. Pour évaluer la performance d'une solution, différents critères peuvent être considérés dans une approche robuste. On peut par, exemple, utiliser le critère de robustesse absolue dit encore critère minimax, le critère de robustesse devié ou le critère de robustesse relative.

Soient S l'ensemble de tous les jeux de valeurs et T l'ensemble de toutes les solutions possibles. Soit $f(x,r)$ la valeur de la fonction objectif associée à la solution $x \in T$ et $r \in S$. Pour un jeu de valeurs donné r, le problème considéré est $min_{x \in T} f(x,r)$.

Une solution est dite robuste absolue si sa performance est la meilleure dans le plus mauvais cas. Elle est définie par :

$$Z_A = min_{x \in T} max_{r \in S} f(x,r)$$

On cherche à minimiser le maximum des valeurs d'une solution sur l'ensemble des solutions possibles. Ce critère est considéré s'il faut prendre une décision avec une extrème prudence.

Les critères qui suivent cherchent à minimiser le regret maximum devié ou relatif. Pour un jeu de valeurs r donné, le regret dévié d'une solution x est l'écart entre sa performance et la performance optimale. Dans le cas du regret relatif, l'écart est rapporté à la performance optimale. Le regret maximum d'une solution x est calculé sur l'ensemble des jeux de valeurs.

Une solution est dite robuste deviée si elle minimise le regret maximum dévié et elle est définie par :

$$Z_D = min_{x \in T} max_{r \in S} \left(f(x,r) - f^*(r)\right)$$

où $f^*(r) = min_{x \in T} f(x,r)$

On cherche à minimiser le regret dévié maximal, c'est-à-dire, l'écart maximal sur tout le voisinage entre la performance de la solution calculée et la performance optimale de chaque jeu de valeurs.

Une solution est dite robuste relative si elle minimise le regret maximum relatif.

$$Z_R = min_{x \in T} max_{r \in S} \frac{(f(x,r) - f^*(r))}{f^*(r)}$$

On cherche à minimiser le regret relatif maximal, c'est-à-dire, le pourcentage de l'écart maximal entre sa performance et la performance optimale de chaque jeu de valeurs.

Ces deux derniers critères constituent un benchmark des performances de décisions vis-à-vis de la meilleure décision possible, sur tout l'ensemble de jeux de valeurs.

Nous avons retenu dans ce chapitre seulement le premier critère c'est-à-dire le critère de robustesse absolue du problème. Les autres critères peuvent être traîtés de la même façon.

1.3 Problème du plus court chemin avec robustesse absolue

Nous proposons dans cette partie de résoudre le problème du plus court chemin robuste absolu "Absolute Robust Shortest Path" (ARSP), qui consiste à trouver parmi tous les chemins possibles celui qui minimise la longueur du chemin pour tous les scénarios, par la programmation linéaire en nombre mixtes. Dans le premier paragraphe de cette partie, nous donnons un exemple d'un plus court chemin robuste absolu puis la formulation du problème. Ensuite, nous présentons un algorithme proposé dans la littérature et nous montrons, par des résultats numériques, qu'il n'est efficace que pour des instances de petite taille. Enfin, nous présentons notre approche et des expérimentations numériques montrant son efficacité pour deux types de graphes.

1.3.1 Exemple

Nous présentons, ci-dessous, une instance du problème ARSP avec deux scénarios. Le graphe est décrit par la figure 1.1. À chaque arc est associé deux coûts : un coût pour le scénario 1 et un autre pour le scénario 2. On s'intéresse au chemin optimal de v_1 à v_4.

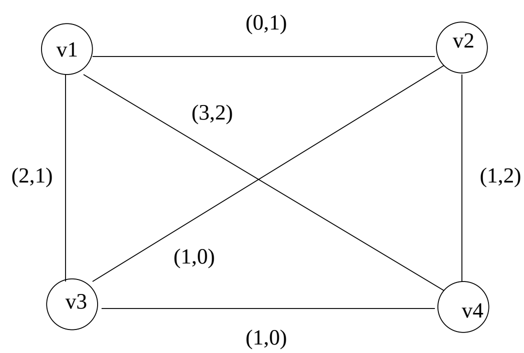

FIG. 1.1 – Exemple d'un graphe à 2 scénarios

Les cinq chemins acycliques qui existent de v_1 à v_4 ainsi que leur longeur sont donnés ci-dessous :

$\pi_1 = v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4$, $long(\pi_1)$ pour le scénario $1 = 2$, $long(\pi_1)$ pour le scénario $2 = 1$;

$\pi_2 = v_1 \rightarrow v_3 \rightarrow v_2 \rightarrow v_4$, $long(\pi_2)$ pour le scénario 1= 4, $long(\pi_2)$ pour le scénario $2 = 3$;

$\pi_3 = v_1 \rightarrow v_2 \rightarrow v_4$, $long(\pi_3)$ pour le scénario 1= 1, $long(\pi_3)$ pour le scénario 2 =3 ;

$\pi_4 = v_1 \rightarrow v_3 \rightarrow v_4$, $long(\pi_4)$ pour le scénario $1 = 3$, $long(\pi_4)$ pour le scénario $2 = 1$;

$\pi_5 = v_1 \rightarrow v_4$, $long(\pi_5)$ pour le scénario $1 = 3$, $long(\pi_5)$ pour le scénario $2 = 2$;

On en déduit que le chemin optimal est le chemin $\pi_1 = v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4$ avec une valeur optimale égale à 2.

1.3.2 Formulation du problème

Soit $G = (V, A)$ un graphe orienté et S un ensemble de scénarios. Soit $g_r(\pi)$ la longueur associée à un chemin π de G avec le scénario $r \in S$. Soit T l'ensemble de tous les chemins possibles de v_1 à v_n dans le graphe G. On cherche le chemin $\pi \in T$ qui minimise la valeur $max\,\{g_r(\pi) : r \in S)\}$. On peut ainsi formuler le problème ARSP comme suit :

$$(ARSP) : min\,\{max\,\{g_r(\pi) : r \in S)\} : \pi \in T\}$$

Une formulation classique en 0-1 du problème est la suivante :
Soit $X \subset \{0,1\}^{|A|}$ l'ensemble des vecteurs caractéristiques associés à l'ensemble de chemins T.

$$X = \left\{ x_{ij} \in \{0,1\} : \left| \begin{array}{l} \displaystyle\sum_{j|(v_i,v_j)\in A} x_{ij} - \sum_{k|(v_k,v_i)\in A} x_{ki} = 0 \ (i = 2, ..., n-1) \\ \displaystyle\sum_{j|(v_1,v_j)\in A} x_{1j} = 1 \\ \displaystyle\sum_{k|(v_k,v_n)\in A} x_{kn} = 1 \end{array} \right. \right\}$$

$g_r(\pi) = \displaystyle\sum_{(i,j)|(v_i,v_j)\in A} c^r_{ij} x_{ij}$ où x est le vecteur caractéristique du chemin π. Pour tout $x \in X$, soit $w(x) = max\,\{g_r(\pi) : r \in S\}$. On obtient le programme en 0-1 suivant :

$$(ARSP_{0-1}) \ : \ min\,\{w(x) : x \in X\}$$

En général, les problèmes minimax sont NP-difficiles et durs à résoudre en pratique. Deux algorithmes ont été proposés dans la littérature pour résoudre ce problème. Yu et Yang [Yu et al., 1998] ont développé un algorithme pseudo-plolynomial que nous décrivons ci-dessous. Un deuxième algorithme, de type branch-and-bound a été proposé dans [Kouvelis et al., 1997]. Les bornes inférieure et supérieure de ce dernier sont obtenues en utilisant les techniques de relaxation agrégéé. Montemanni et al. [Montemanni et al., 2003] ont aussi traité ce problème mais en caractérisant l'incertitude des données par des intervalles. Ils ont proposé un algorithme branch-and-bound fondé sur une nouvelle borne inférieure.

1.3.3 Algorithme de Yu et Yang

Pour la résolution de ARSP, Yu et Yang ont proposé dans [Yu et al., 1998] un algorithme de complexité pseudo-polynomiale (cf Alg 1).

Algorithm 1 Algorithm de Yu et Yang [Yu et al., 1998]

Début

{initialisation}

trier les coûts des arcs pour déterminer les $I^r(A)$ *et calculer* $P_r = \sum_{a \in I^r(A)} c_a^r,\ r \in S$;
Calculer $P_{max} = max_{r \in S}\, P_r$;
Pour *tout arc* $(v, v') \notin A$***, soit*** $c_{vv'}^r = P_{max},\ \forall r \in S$ ***;***
soit $c_{vv'}^r = 0, \forall v \in V\, et\, r \in S$;
Pour $\alpha_1 = 0$ *jusqu'à* P_1 ***faire***
....
Pour $\alpha_{|S|} = 0$ *jusqu'à* $P_{|S|}$ ***faire***
Pour $v \in V$ ***faire***
$f_v^1\left(\alpha_1,\, ...,\, \alpha_{|S|}\right) = min\left\{max_{r \in S}\left\{\alpha_r + c_{vn}^r\right\},\, P_{max}\right\}$;
Pour $\tau = 2$ *jusqu'à* $|V| - 1$ ***faire***
Pour $v \in V$ ***faire***
Pour $\alpha_1 = 0$ *jusqu'à* P_1 ***faire***
....
Pour $\alpha_{|S|} = 0$ *jusqu'à* $P_{|S|}$ ***faire***
$f_v^\tau\left(\alpha_1,\, ...,\, \alpha_{|S|}\right) = P_{max}$;
Pour $v^{'} \in V$ ***faire***
Si $\alpha_r + c_{vv'}^r \leq P_r, \forall\, r \in S$ et $f_{v'}^{\tau-1}\left(\alpha_1 + c_{vv'}^1,\, ...,\, \alpha_{|S|} + c_{vv'}^{|S|}\right) < f_v^\tau\left(\alpha_1,\, ...,\, \alpha_{|S|}\right)$
alors
$f_v^\tau\left(\alpha_1,\, ...,\, \alpha_{|S|}\right) = f_{v'}^{\tau-1}\left(\alpha_1 + c_{vv'}^1,\, ...,\, \alpha_{|S|} + c_{vv'}^{|S|}\right)$;
finsi
$f_{v_1}^{|V|-1}\left(\alpha_1,\, ...,\, \alpha_{|S|}\right)$ est la solution optimale.
finpour

Fin

Il est fondé sur le principe suivant :

Soit $G = (V,\, A)$ un graphe. À chaque arc $a \in A$ sont associés $|S|$ coûts positifs c_a^r. Les c_a^r étant positifs, un chemin optimal robuste est un chemin acyclique qui contient au plus $(|V|-1)$ arcs. Ainsi, nous pouvons limiter la recherche à ces chemins. La valeur de la fonction objectif pour tous les chemins acycliques de s à t est inférieure ou égale à $P_r = \sum_{a \in I^r(A)} c_a^r$, où $I^r(A)$

désigne l'ensemble de $(|V|-1)$ arcs ayant les c_a^r les plus grands pour le scénario $r \in S$. Ainsi, $P_{max} = max_{r\in S} P_r$ est une borne supérieure de la solution optimale. L'algorithme est fondé sur la programmation dynamique. On pose : $f_v^\tau\left(\alpha_1, ..., \alpha_{|S|}\right)$ la longueur maximale pour le chemin π, $max_{r\in S}\, g_r(\pi)$ où π correspond au chemin minimisant cette valeur sur l'ensemble des chemins possibles de s à t, possédant des longueurs partielles de s à v égales aux α_r et contenant au plus τ arcs de v à t.

Une condition initiale est définie ainsi : $f_v^1\left(\alpha_1, ..., \alpha_{|S|}\right) = min\{max_{r\in S}\{(\alpha_r + c_{vt}^r\}, P_{max}\}$. Par récursivité, on aura : $f_v^\tau\left(\alpha_1, ..., \alpha_{|S|}\right) = min_{v'\in V}\{f_{v'}^{\tau-1}\left(\alpha_1 + c_{vv'}^1, ..., \alpha_{|S|} + c_{vv'}^{|S|}\right)\}\forall \tau = 2, ..., |V|-1$. Étant donné qu'un chemin optimal contient au plus $|V|-1$ arcs alors $\tau \leq |V|-1$. La solution optimale est donnée par $f_s^{|V|-1}(0, ..., 0)$.

La complexité en temps de cet algorithme est en $O\left(|V|^3\left(P_{max}\right)^{|S|}\right)$. C'est un algorithme pseudo-polynomial pour un nombre de scénarios fixé puisque P_{max} depend des coûts des arcs.

Cet algorithme est présenté dans [Yu et al., 1998] sans résultats numériques. Afin de tester son efficacité, nous l'avons implémenté en langage C sur une machine à processeur alpha à 333 MHz. Les jeux d'essais ont été générés aléatoirement, les coefficients c_a^r sont distribués uniformément sur [0, 10]. La taille des instances considérées varie entre 10 et 50 nœuds avec 2 et/ou 4 scénarios. Les densités des graphes considérés sont comprises entre 15% et 85%. Les résultats sont indiqués dans le tableau 1.1. Chaque ligne du tableau représente la moyenne des résultats obtenus sur 10 instances.

Le tableau 1.1 montre qu'il est difficile de traiter des grandes instances avec l'algorithme de [Yu et al., 1998]. Le temps de calcul est relativement élevé et l'espace memoire nécessaire est très important. Par exemple, pour une instance de 50 nœuds et 2 scénarios, le temps de calcul dépasse 3 heures. Dans le cas de 4 scénarios, il n'est pas possible de résoudre des instances de plus de 10 nœuds faute d'espace mémoire. Il est à noter que si les coéfficients c_a^r sont choisis entre [0, 100], alors aucune instance de 10 nœuds et 2 scénarios ne peut être résolue. Par ailleurs, il serait sans doute possible d'augmenter la taille des instances pouvant être traitées par cet algorithme en utilisant une machine plus puissante avec une mémoire plus importante. Néanmoins, quelle que soit la machine utilisée, l'intérêt pratique de cet algorithme reste limité.

Dans la section suivante, nous proposons une formulation linéaire en 0-1 pour le problème ARSP que l'on appellera Lin-ARSP, puis, nous présentons des résultats numériques qui montrent que cette approche permet de résoudre de grandes instances du problème quel que

Nb_nœuds	Densité	Nb_scénarios	Temps_CPU	Nb_instances
10	85%	2	2.4	10
	75%		2.03	10
	50%		1.76	10
	25%		0.85	10
	15%		0.7	10
30	85%	2	864.2	10
	75%		811.1	10
	50%		740.4	10
	25%		588.2	10
	15%		436.6	10
50	85%	2	11918.10	10
	75%		11025.7	10
	50%		10345.0	10
	25%		8968.4	10
	15%		7546.3	10
10	85%	4	-	0
	75%		-	0
	50%		1480.0	10
	25%		363.8	10
	15%		346.3	10

Tab. 1.1 – Résultats numériques obtenus par l'algorithme de [Yu et al., 1998]
- : la solution n'est pas obtenue pour ce type d'instances

soit le nombre de scénarios.

1.3.4 Formulation du problème par un programme linéaire en variables mixtes

Il est facile de mettre le programme ARSP sous une forme linéaire :

$$(Lin - ARSP) \begin{cases} Min\, \alpha & \\ \sum_{j|(v_1,v_j)\in A} x_{1j} = 1 & (1) \\ \sum_{k|(v_k,v_n)\in A} x_{kn} = 1 & (2) \\ \sum_{j|(v_i,v_j)\in A} x_{ij} - \sum_{k|(v_k,v_i)\in A} x_{ki} = 0 \quad i = 2, ..., n-1 & (3) \\ \sum_{(i,j)|(v_i,v_j)\in A} c_{ij}^r x_{ij} - \alpha \leq 0 \qquad r = 1, ..., |S| & (4) \\ x_{ij} \in \{0,1\} \quad ((v_i, v_j) \in A) & (5) \end{cases}$$

En considérant l'objectif et la contrainte (4), il est facile de vérifier qu'à l'optimum, la valeur de α sera égale à $min \left\{ max \left\{ \sum_{(i,j)|(v_i,v_j)\in A} c_{ij}^r x_{ij} : r \in S \right\} : x \in X \right\}$.

L'intérêt principal de cette formulation réside dans le peu de variables et de contraintes ajoutées par rapport à la formulation initiale. Nous avons en effet ajouté une seule variable α et r contraintes, chacune de ces contraintes correspondant à un scénario.

1.3.5 Expérimentations

Pour évaluer les performances de notre approche, nous avons effectué des tests numériques sur différentes instances. Le programme (Lin-ARSP) a éte résolu en utilisant CPLEX 6.0 [Cplex, 1998] sur un ordinateur de type Pentium II à 300 MHz. Les expérimentations ont été réalisées sur des instances générées aléatoirement. Les coéfficients c_a^r sont des entiers uniformément distribués entre 0 et 10. Le tableau 1.2 résume les résultats numériques concernant des problèmes de 50 et 100 nœuds avec une densité qui varie de 15% à 85%, et un nombre de scénarios égal à 10, 20 et 30. Chaque ligne du tableau représente la moyenne sur 10 instances. Pour chaque problème, nous indiquons le nombre de nœuds (Nb_nœuds), la densité (Densité), le nombre de scénarios (Nb_Scénarios), la durée totale de résolution (Temps_CPU) et le nombre d'instances résolues (Nb_instances).

Nb_nœuds	Densité	Nb_scénarios	Temps_CPU	Nb_instances
50	85%	10	4.45	10
	75%		1.3	10
	50%		1.9	10
	25%		1.22	10
	15%		0.72	10
50	85%	20	7.44	10
	75%		4.19	10
	50%		5.61	10
	25%		3.09	10
	15%		2.72	10
50	85%	30	12.29	10
	75%		10.04	10
	50%		8.27	10
	25%		5.44	10
	15%		4.73	10
100	85%	10	52.67	10
	75%		62.91	10
	50%		29.58	10
	25%		18.72	10
	15%		21.72	10
100	85%	20	134.76	10
	75%		104.93	10
	50%		127.58	10
	25%		51.87	10
	15%		70.41	10
100	85%	30	339.71	10
	75%		126.77	10
	50%		211.93	10
	25%		98.16	10
	15%		146.94	10

TAB. 1.2 – Résultats numériques de Lin-ARSP

Le tableau 1.2 montre que notre approche pour résoudre le problème ARSP est très efficace. En effet, nous pouvons résoudre de grandes instances quel que soit le nombre de scénarios. Le temps de calcul ne dépasse pas 340 secondes en moyenne pour un graphe dense de 100 nœuds. On peut noter que, contrairement à l'algorithme de [Yu et al., 1998] qui est très sensible au nombre de scénarios, le temps de calcul nécessaire pour Lin-ARSP ne croît pas beaucoup avec le nombre de scénarios.

Afin de comparer notre approche vis à vis de l'algorithme branch-and-bound proposé dans [Kouvelis et al., 1997], nous avons considéré le même type de graphes que ces auteurs : des graphes en niveaux (bipartis en plusieurs niveaux) (voir FIG. 1.2). Un graphe en m niveaux est défini comme suit : l'ensemble de nœuds $V = \{v_1, ..., v_n\}$ est subdivisé en des sous ensembles disjoints $\{v_1\}$, V_1, V_2, ..., V_m, $\{v_n\}$. Les arcs existent seulement de v_1 à V_1, de V_m à v_n et de V_k à V_{k+1} pour $k = 1, ..., m - 1$. Celà nous permet d'eviter le cas de solutions avec un petit nombre d'arcs.

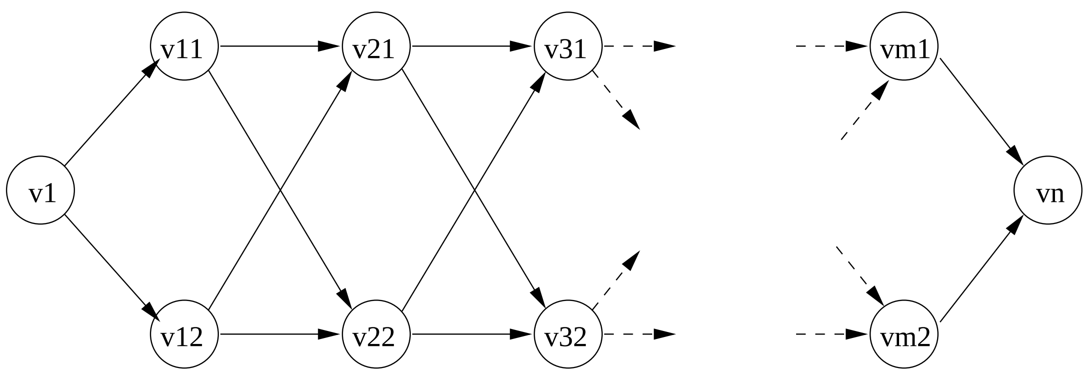

FIG. 1.2 – Exemple d'un graphe en m niveaux de largeur 2

Le tableau 1.3 présente une étude comparative entre Lin-ARSP et la procédure de [Kouvelis et al., 1997]. Les expérimentations de [Kouvelis et al., 1997] ont été réalisées sur une machine de type VAX6400 et les expérimentations de Lin-ARSP ont été menées sur un PC Pentium II à 300MHz. Les performances de ces deux machines sont comparables. Nous avons réalisé les tests avec des graphes de 20, 30, 40, 50, 100 et 500 nœuds, et pour un nombre de scénarios variant de 10 à 40. Chaque graphe est composé de m niveaux et chaque niveau est composé de 4 nœuds. Tous les arcs possibles sont présents. Les coéfficients c_a^r sont des entiers uniformément distribués entre 0 et 100. Pour chaque type de graphe (nombre de nœuds et nombre de scénarios), nous avons généré 10 instances. Pour Lin-ARSP, nous indiquons le temps de résolution (Temps_CPU) et le nombre d'instances résolues (Nb_instances) et, pour la procédure de [Kouvelis et al., 1997], nous indiquons le temps de résolution (Temps_CPU). Le tableau1.3 confirme l'efficacité de Lin-ARSP. Lin-ARSP est beaucoup plus rapide que la procédure branch-and-bound de [Kouvelis et al., 1997]. Par exemple, la résolution d'une instance de 40 nœuds et 30 scénarios par Lin-ARSP nécessite moins d'une seconde alors qu'avec la procédure de [Kouvelis et al., 1997] prend plus de 400 secondes. Avec Lin-ARSP, nous arrivons à résoudre rapidement des grandes instances : instances à 500 nœuds et 40 scénarios en moins de 600 secondes.

Données		Lin-ARSP		Kouvelis et Yu
Nb_nœuds	Nb_scénarios	Temps_CPU	Nb_instances	Temps_CPU
20	10	0.025	10	24.0
	20	0.048	10	39.3
	30	0.057	10	102.9
30	10	0.048	10	103.4
	20	0.071	10	144.7
	30	0.096	10	161.2
40	10	0.086	10	269.3
	20	0.115	10	313.2
	30	0.171	10	425.2
50	10	0.11	10	-
	20	0.183	10	-
	30	0.254	10	-
	40	0.332	10	-
100	10	0.545	10	-
	20	0.834	10	-
	30	1.877	10	-
	40	1.502	10	-
150	10	1.823	10	-
	20	3.238	10	-
	30	4.236	10	-
	40	5.748	10	-
500	10	142.035	10	-
	20	332.385	10	-
	30	386.09	10	-
	40	537.181	10	-

TAB. 1.3 – Comparaison entre Lin-ARSP et le branch-and-bound de [Kouvelis et al., 1997]

1.4 Conclusion

Nous avons considéré dans ce chapitre le problème du plus court chemin avec incertitude des données. Une approche par scénarios a été adoptée pour caractériser les incertitudes. Nous nous sommes intéressés plus particulièrement au problème du plus court chemin robuste absolu "Absolute Robust Shortest Path" (ARSP) qui consiste à trouver, parmi tous les chemins possibles, celui qui minimise la longueur du chemin pour tous les scénarios. Nous avons proposé de le résoudre en utilisant un outil de programmation linéaire mixte. Cette approche présente plusieurs avantages. Elle est très facile à implémenter, compétitive et même supérieure à d'autres méthodes de la littérature. Elle nous a permis de résoudre de manière très efficace des grandes instances. Notons également qu'il est très facile d'ajouter des contraintes supplémentaires à notre problème initial sans faire de modifications majeures dans notre programme ce qui n'est pas le cas pour d'autres méthodes .

Bibliographie

[Ahuja, 1997] R.K. Ahuja, 'Flows and paths', Annotated bibliogaphies in combinatorial optimization, Edited by M.Dell'Amico, F. Maffioli and S. Martello, Wiley, p. 283-309 (1997).

[Billionnet et al., 2002] A. Billionnet, K. Djebali, 'Integer linear programming for the robust shortest path problem', en soumission à Computational Optimization and Applications, (2002) (en révision).

[Cplex, 1998] Ilog, Inc., Cplex Division, 'Using the Cplex callable library', Version 6.0 (1998).

[Daniels et al., 95] R.L. Daniels, P. Kouvelis, 'Robust scheduling to hedge against processing time uncertainty in single stage production', Management Science, Vol. 41, p. 363-376 (1995).

[Dijkstra, 1959] E.W. Dijkstra, 'Note on two problems in connection with graphs', Numerishe Mathematik, number 1, p. 269-271 (1959).

[Deodhare et al., 1998] D. Deodhare, M. Vidyasagar, S. Keerthi, 'Synthesis of fault-Tolerant feedforward neural networks using minimax optimization', IEEE Transactions on Neural Networks, Vol. 9, Nř 5, p. 891-900 (1998).

[Hermann, 1999] J. W. Hermann, 'A genetic algorithm for a minimax network design problem' Research Report, Departement of Mechanical Engineering, University of Maryland (1999).

[Gupta et al., 1972] S.K. Gupta, J. Rosenhead, 'Robustness in sequential investment decisions', management science, Vol. 15, p. 18-29 (1972).

[Kouvelis et al., 1992] P. Kouvelis, A.A. Karawarwala, G.J. Guiterez, 'Algorithms for robust single and multiple period layout planning for manufacturing systems', European Journal of Operational Research, Vol. 63, p. 287-303 (1992).

[Kouvelis et al., 1997] P. Kouvelis, G. Yu, 'Robust discrete optimization and its applications', Kluwer Academic Publishers, Norwell, MA (1997).

[Montemanni et al., 2003] R. Montemanni, L.M. Gambardella, A.V. Donati, 'A branch and bound algorithm for the robust shortest path problem with interval data', Oper. Res. Letter, à paraître.

[Rosenhead et al., 1972] M.J. Rosenhead, M. Elton, S.K. Gupta, 'Robustness and optimality as criteria for strtegic decisions', Operational Research Quarterly, Vol. 23, p. 413-430 (1972).

[Sengupta, 1991] J.K. Sengupta, 'Robust decisions in economic models', Computers and Operations Research, Vol. 18, p. 221-232 (1991).

[Veeravalli et al., 1994] V.V. Veeravalli, H.V. Poor, 'Minimax robust decentralizes detection', IEEE Transactions on Information Theory, Vol. 40, Nř 1, p. 35-40 (1994).

[Yu et al., 1998] G. Yu, J. Yang, 'On the robust shortest path problem', Computers and Operations Research, Vol. 25, Nř 6, p. 457-468 (1998).

Chapitre 2

Optimisation de portefeuilles

2.1 Introduction

Ce chapitre se situe dans le cadre de la théorie du portefeuille. Nous considérons le problème de la sélection optimale de titres parmi un ensemble de titres disponibles en utilisant le modèle de Markowitz. Ce problème, sous sa forme classique, est un problème d'optimisation quadratique convexe pour lequel il existe de nombreuses méthodes permettant de le résoudre de façon exacte. En pratique, certaines contraintes supplémentaires sont à prendre en compte dans la réalité, ce qui rend le problème NP-difficile. Nous étudions dans ce chapitre deux problèmes de portefeuille quadratiques non convexes. Le premier consiste à constituer un portefeuille avec une contrainte sur le nombre de titres à sélectionner, plus précisément le problème de la construction de la frontière d'efficience d'un portefeuille sous cette contrainte. Le deuxième problème concerne le rééquilibrage d'un portefeuille avec des coûts de transactions fixes et linéaires.

Dans la première partie de ce chapitre, nous présentons le problème classique de sélection d'un portefeuille optimal. Nous commençons par la définition du problème et la présentation des différentes modélisations proposées dans la littérature. Ensuite, nous passons au problème de choix d'un portefeuille optimal.

Dans la deuxième partie, nous étudions le cas du problème avec une contrainte de cardinalité que nous formulons comme un programme quadratique en variables mixtes. Nous proposons pour ce problème une heuristique fondée sur la relaxation continue (convexe) du problème et sur une fixation des variables. Nous présentons des résultats de calcul permettant de comparer les performances de notre heuristique avec d'autres heuristiques développées dans la littérature.

Dans la troisième partie, nous considérons le problème de rééquilibrage d'un portefeuille. Deux cas qui diffèrent par la fonction exprimant les coûts de transactions seront étudiés. Nous adaptons l'heuristique proposée dans la deuxième partie pour résoudre le problème avec des coûts de transactions fixes et linéaires.

2.2 Problème classique de sélection d'un portefeuille optimal

Le problème de sélection d'un portefeuille, sous sa forme classique, consiste à choisir de façon optimale un sous ensemble de titres parmi n titres disponibles. Le rendement procuré par chaque titre est une variable aléatoire dont on suppose que l'on connaît l'espérance et la variance. On cherche à sélectionner un sous ensemble de titres qui minimise le risque et maximise l'espérance du revenu. Un modèle de base où le risque est mesuré par la variance du rendement a été proposé par H. Markowitz dans son ouvrage Portfolio Selection [Markowitz, 1952] dit Modèle "Espérance-Variance" ou "Moyenne-Variance". Nous avons considéré ce modèle de base dans notre travail. On trouvera, ci-après, la description de ce modèle ainsi que d'autres modèles proposés dans la littérature.

2.2.1 Modèle "Espérance-Variance"

Dans ce modèle, le rendement espéré a été utilisé comme mesure du rendement et la variance (ou l'ecart-type) comme mesure du risque. Avant de présenter ce modèle, nous définissons, ci-dessous, les notations associées aux différentes données du problème.
$\{t_1, t_2, ..., t_n\}$: ensemble des titres disponibles ;
E_i : rendement espéré procuré par le titre t_i, $i = 1, ..., n$;
Var_i : variance du rendement associé au titre t_i, $i = 1, ..., n$;
Cov_{ij} : covariance des rendements associés aux deux titres t_i et t_j ;
ρ : espérance du rendement que doit procurer le portefeuille retenu.

À chaque titre t_i est associée une variable x_i, $0 \leq x_i \leq 1$. Elle représente la proportion de la valeur totale du portefeuille investie dans le titre t_i. Dans le modèle de Markowitz, on cherche à minimiser le risque du portefeuille exprimé par la variance du rendement, tout en imposant une certaine espérance de ce rendement. On peut formuler ce modèle par le programme mathématique $(QP1)$ qui consiste à minimiser une fonction quadratique convexe soumise à des contraintes linéaires :

$$(QP1)\begin{cases} Min & \sum_{i=1}^{n}\sum_{j=1}^{n} Cov_{ij}\, x_i x_j \\ s.c. & \sum_{i=1}^{n} x_i = 1 & (1) \\ & \sum_{i=1}^{n} E_i x_i = \rho & (2) \\ & x_i \in [0,\, 1] \; i = 1, ..., n & (3) \end{cases}$$

La première contrainte impose que la somme des proportions des titres soit égale à 1 et la deuxième contrainte permet d'assurer un niveau donné d'espérance de rendement. On peut également imposer, pour chaque titre t_i, une proportion minimale LB_i et une proportion maximale UB_i dans le portefeuille. On obtient alors le modèle suivant.

$$(QP1')\begin{cases} Min & \sum_{i=1}^{n}\sum_{j=1}^{n} Cov_{ij}\, x_i x_j \\ s.c. & \sum_{i=1}^{n} x_i = 1 & (1) \\ & \sum_{i=1}^{n} E_i x_i = \rho & (2) \\ & LB_i \leq x_i \leq UB_i \; i = 1, ..., n & (4) \end{cases}$$

D'autres modèles ont été développés dans la littérature. Nous décrivons ici quatre d'entre eux qui nous paraissent importants.

2.2.2 Modèle "Mean-absolute deviation"

Konno et Yamazaki [Konno et al., 1991] proposent un modèle appelé "Mean Absolute Deviation" dans lequel le risque est mesuré par la déviation absolue moyenne des rendements. Soit $E_i = \dfrac{1}{T}\sum_{t=1}^{T} r_{it}$, l'espérance du rendement du titre t_i $(i = 1, ..., n)$ où r_{it} est le rendement du titre t_i durant la période $t\,(t = 1, ..., T)$, et T le nombre de périodes considérées. En ce qui concerne l'espérance du rendement du portefeuille, comme dans le cas du modèle de Markowitz, elle doit être égale à une valeur ρ fixée à l'avance. La formulation obtenue est donnée par :

$$\begin{cases} Min & \dfrac{1}{T}\displaystyle\sum_{t=1}^{T}\left|\sum_{i=1}^{n}(r_{it}-E_i)x_i\right| \\ s.c. & \displaystyle\sum_{i=1}^{n}x_i=1 \\ & \displaystyle\sum_{i=1}^{n}E_ix_i=\rho \\ & x_i\in[0,\,1]\ i=1,\,...,\,n \end{cases}$$

Ce modèle a été aussi considèré par Speranza dans [Speranza, 1996] mais seulement avec des déviations négatives. Le modèle obtenu est le suivant :

$$\begin{cases} Min & \dfrac{1}{T}\displaystyle\sum_{t=1}^{T}y_t \\ s.c. & \displaystyle\sum_{i=1}^{n}x_i=1 \\ & \displaystyle\sum_{i=1}^{n}E_ix_i=\rho \\ & y_t+\sum_{i=1}^{n}(r_{it}-E_i)x_i\geq 0 \quad t=1,\,...,\,T \\ & x_i\in[0,\,1] \quad\quad i=1,\,...,\,n \end{cases}$$

Contrairement aux suggestions de Konno et Yamazaki pour ce modèle qui ne tient pas compte de la matrice de covariance, Simaan [Simaan, 1997] montre que l'ignorance de cette matrice agit beaucoup sur les bénéfices. L'intérêt principal de ce modèle est sa linéarité.

2.2.3 Modèle minimax

Young [Young, 1998] considère également un horizon de T périodes et propose une approche minimax pour modéliser le problème. Étant donné r_{it}, le rendement du titre i durant la période t, on déduit facilement $E_i=\dfrac{1}{T}\displaystyle\sum_{t=1}^{T}r_{it}$, l'espérance du rendement du titre i sur l'ensemble des périodes. Dans ce modèle, on cherche à maximiser M_p, le rendement du portefeuille au cours de la période où ce dernier est le plus faible. Le rendement du portefeuille au cours de la période t est égal à $\sum_{i=1}^{n}x_ir_{it}$; on obtient alors le modèle suivant.

$$\begin{cases} Max & M_p \\ s.c. & \sum_{i=1}^{n} x_i r_{it} - M_p \geq 0 \quad t = 1, ..., T \\ & \sum_{i=1}^{n} x_i = 1 \\ & \sum_{i=1}^{n} E_i x_i = \rho \\ & x_i \in [0, 1] \quad\quad\quad i = 1, ..., n \end{cases}$$

2.2.4 Modèle diagonal de Sharpe

Afin de réduire le nombre de paramètres à estimer (les n espérances de rendement, les n variances et les $n(n-1)/2$ covariances), W. Sharpe [Sharpe, 1963] a proposé en 1963, un nouveau modèle dit "Modèle diagonal" ou "Modèle à un facteur" où le seul facteur de risque commun à tous les titres est le portefeuille du marché.

Soit $R_i = \alpha_i + \beta_i R_M + e_i$, $i = 1, ..., n$, le taux de rendement associé au titre i, où α_i et β_i sont des constantes propres au titre i, R_M est le taux de rentabilité aléatoire du portefeuille du marché et e_i une perturbation aléatoire d'espérance nulle et indépendante de la rentabilité du marché et des perturbations affectant les autres titres. On en déduit alors que E_i, l'espérance de rendement, est égale à $(\alpha_i + \beta_i E(R_M))$, que Var_i, la variance de rendement, est égale à $\left(\beta_i^2 \sigma_M^2 + \sigma_{e_i}^2\right)$ et que Cov_{ij}, la covriance entre les rendements de deux titres i et j, est égale à $\left(\beta_i \beta_j \sigma_M^2\right)$. Le modèle obtenu est le suivant :

$$\begin{cases} Min & \sum_{i=1}^{n} \sigma_{e_i}^2 x_i^2 + \sum_{i=1}^{n}\sum_{j=1}^{n} \beta_i \beta_j \sigma_M^2 x_i x_j \\ s.c. & \sum_{i=1}^{n} x_i = 1 \\ & \sum_{i=1}^{n} E_i x_i = \rho \\ & x_i \in [0, 1] \; i = 1, ..., n \end{cases}$$

Plus tard, des travaux empiriques ont mis en évidence l'existence d'autres facteurs globaux économiques à prendre en compte, outre le marché lui même. Ainsi, le modèle a été généralisé pour le cas multifacteurs (voir [Rosenberg, 1974]).

2.2.5 Le choix d'un portefeuille et la frontière d'efficience

Pour le choix d'un portefeuille optimal, un investisseur doit tout d'abord déterminer quels sont les portefeuilles efficaces. Ensuite, en fonction de sa richesse initiale et de son attitude personnelle vis-à-vis du risque, c'est-à-dire son degré de prudence, il choisira le portefeuille qui réalisera pour lui le meilleur compromis entre risque et rentabilité espérée. Un portefeuille est efficace par rapport à un ensemble de portefeuilles si, pour un niveau de risque donné (variance du rendement), il a la plus forte espérance de rendement et si, pour un niveau donné d'espérance de rendement, il a le risque le plus faible. La frontière d'efficience correspond à l'ensemble des portefeuilles efficaces. Cela revient à considérer une courbe dans le plan Espérance-Risque, appelée *frontière d'efficience d'un portefeuille,* regroupant l'ensemble de ces portefeuilles efficaces. La résolution de $(QP1)$ pour différentes valeurs de ρ permet de tracer cette courbe en déterminant différents points. Cette courbe est convexe et continue pour le cas classique du problème de portefeuille (Modèle $QP1$). La figure 2.1 en donne une illustration. Nous la désignerons, dans la suite de ce travail par *Frontière d'Efficience Classique* (FEC).

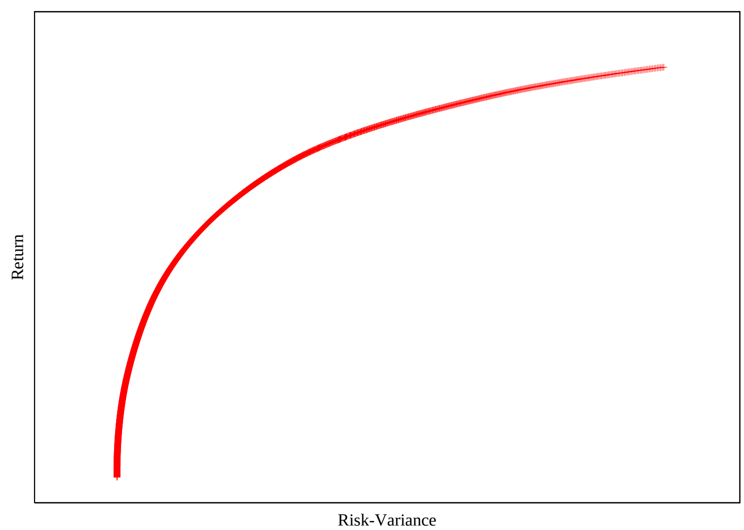

FIG. 2.1 – Frontière d'Efficience Classique (FEC)

Il existe plusieurs algorithmes efficaces permettant de résoudre $(QP1)$ de façon exacte, il est donc facile de tracer la courbe (FEC) point par point. Cependant, comme nous l'avons déjà évoqué, certaines contraintes supplémentaires sont à prendre en compte dans les problèmes réels de gestion de portefeuille. On retrouve par exemple :

1. *des contraintes de seuil* : afin d'éviter d'avoir des titres avec de trop faibles proportions, on impose à chaque titre t_i, une proportion minimale LB_i et une proportion maximale UB_i. La proportion d'un titre sélectionné devra être comprise entre ces deux bornes sinon cette proportion sera égale à zéro. Ces contraintes peuvent se modéliser comme suit :

$$y_i LB_i \leq x_i \leq y_i UB_i,\ y_i \in \{0,\, 1\}\ \ (i = 1,\, ...,\, n)$$

   où la variable y_i est une variable bivalente qui vaut 1 si et seulement si le titre i est selectionné.

2. *des contraintes de cardinalité :* afin de contrôler les coûts des transactions, certains investisseurs limitent le nombre de titres à sélectionner. Ceci peut s'exprimer par l'équation suivante :

$$\sum_{i=1}^{n} y_i = K,\ y_i \in \{0,\, 1\}$$

3. *des contraintes de fraction (roundlots) :* ces contraintes imposent d'acheter un titre par un nombre entier de fraction ou portion du portefeuille fixé à l'avance. À chaque titre x_i est associée une fraction f_i exprimant une fonction d'achat ou de vente de ce titre. On peut modéliser ces contraintes comme suit :

$$x_i = f_i y_i,\ y_i \in \mathbb{N}\ (i = 1,\, ...,\, n)$$

Toutes ces contraintes font intervenir des variables discrètes ce qui rend le problème de sélection d'un portefeuille beaucoup plus difficile à résoudre. Nous étudions, dans la section suivante, le problème de la constitution d'un portefeuille optimal comportant seulement K titres choisis parmi n titres disponibles. Nous le formulons comme un programme quadratique en variables mixtes. Ensuite, nous discutons les propriétés de la frontière d'efficience de ce problème puis proposons, pour la tracer, une heuristique fondée sur la résolution de la relaxation continue de ce programme et sur une procédure simple de fixation de variables utilisant la solution de cette relaxation. Enfin, nous présentons des résultats de calcul permettant de comparer notre approche avec d'autres heuristiques présentées dans la littérature.

2.3 Problème de portefeuille avec une contrainte sur le nombre de titres

Ce problème peut se formuler par le programme mathématique $(QP2)$. Pour construire ce modèle, les contraintes (5) et (6) ont été ajoutées au problème initial $(QP1)$. Elles permettent d'imposer la sélection de K titres. La contrainte (6) permet également d'imposer des bornes inférieures et supérieures sur les proportions de chaque titre sélectionné.

$$(QP2)\begin{cases} Min \quad \sum_{i=1}^{n}\sum_{j=1}^{n} Cov_{ij}\, x_i\, x_j & \\ s.c. & \\ \quad \sum_{i=1}^{n} x_i = 1 & (1)\\ \quad \sum_{i=1}^{n} E_i x_i = \rho & (2)\\ \quad \sum_{i=1}^{n} y_i = K & (5)\\ \quad y_i LB_i \leq x_i \leq y_i UB_i,\ i = 1, ..., n & (6)\\ \quad y_i \in \{0, 1\},\ i = 1, ..., n & \end{cases}$$

Comme dans les autres modèles $x_i\,(i = 1, ..., n)$ représente la proportion du titre i dans le portefeuille $(0 \leq x_i \leq 1)$. La variable $y_i\,(i = 1, ..., n)$ est une variable bivalente qui vaut 1 si et seulement si une proportion non nulle du titre i est sélectionnée dans le portefeuille.

$(QP2)$ est un problème quadratique en variables mixtes (non convexe). Il n'existe pas de méthodes efficaces pour le résoudre de façon optimale d'autant plus que dans la pratique un portefeuille peut contenir plusieurs centaines de titres voire plusieurs milliers. Dans ce cas, l'application d'une heuristique est indispensable surtout s'il s'agit de tracer la frontière d'efficience du problème, ce qui nécessite la résolution du problème pour de nombreuses valeurs du rendement espéré. Avant de présenter notre méthode de résolution, nous allons tout d'abord montrer l'effet de la contrainte de cardinalité sur la frontière d'efficience.

2.3.1 Construction de la frontière d'efficience pour le problème de portefeuille avec une contrainte sur le nombre de titres

En présence de la contrainte de cardinalité, la frontière d'efficience devient discontinue. Pour certains niveaux de revenus, il n' y a pas de solutions : il existe des portefeuilles avec un revenu plus élevé et un risque plus petit. Pour bien illustrer cet aspect de discontinuité, considérons l'exemple suivant, présenté dans [Chang et al., 2000], où l'on doit choisir 2 titres parmi 4.

Titre	Espérance de rendement	Ecart-type
1	0.004798	0.046341
2	0.000659	0.035770
3	0.003174	0.030474
4	0.001377	0.035770

TAB. 2.1 – Exemple : Espérance de rendement et écart-type pour les titres

$$\begin{pmatrix} 1 & 0.118368 & 0.143822 & 0.252213 \\ & 1 & 0.164589 & 0.099763 \\ & & 1 & 0.083122 \\ & & & 1 \end{pmatrix}$$

Exemple **:** Matrice de corrélation pour les titres 1,2,3 et 4.

En considérant les 6 combinaisons possibles de deux titres on peut construire les 6 frontières d'efficacité correspondantes (voir FIG. 2.2). Ensuite, en éliminant tous les portefeuilles inefficaces, on obtient la frontière d'efficience du problème dont on voit bien la discontinuité (voir FIG.2.3). Cette frontière sera désignée dans la suite de ce travail par *la Frontière d'Efficience avec Contrainte de Cardinalité* (FECC).

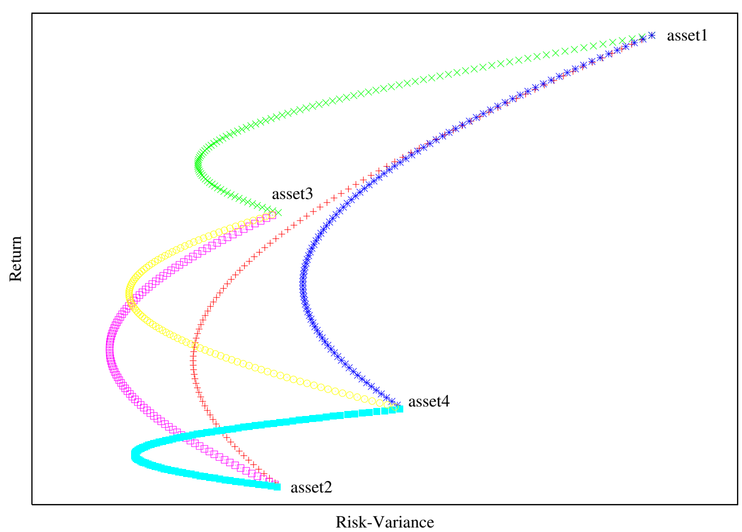

FIG. 2.2 – Frontières d'efficiences à deux titres

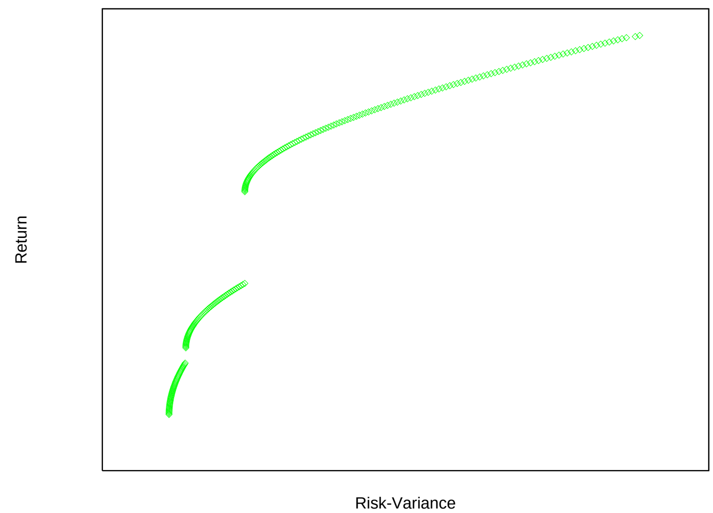

FIG. 2.3 – Frontière d'efficience avec la contrainte de cardinalité (FECC)

Pour illustrer cette discontinuité, nous avons déterminé les meilleurs portefeuilles ayant un rendement de 0.0033988. D'après la table 2.1, nous constatons que ces portefeuilles doivent obligatoirement contenir une certaine proportion du titre t_1. La table 2.2 donne les caractéristiques du portefeuille ayant ce rendement pour chaque paire possible de titres.

Titres composant le portefeuille	Proportion du titre T_1	Proportion du titre T_2	Risque (variance du rendement)
t_1, t_2	0.662	0.338	0.001175
t_1, t_3	0.139	0.861	0.000779
t_1, t_4	0.591	0.409	0.001166

TAB. 2.2 – Exemple : Détermination du portefeuille assurant un rendement égal à 0.0033988 et minimisant le risque

Les résultats de la table 2.2 montrent que le meilleur portefeuille assurant un rendement de 0.0033988 est composé des deux titres t_1 et t_3 en proportions respectives de 0.139 et 0.861. Le risque lié à ce portefeuille, c'est à dire la variance du rendement, est égal à 0.000779. Considérons maintenant le portefeuille formé des deux titres t_1 et t_2 en proportions respectives de 0.7 et 0.3. Le rendement et le risque valent respectivement 0.0036612 et 0.000734. Ceci illustre la discontinuité de la frontière d'efficacité : il n'y a pas de portefeuille efficace ayant un rendement de 0.0033988.

À notre connaissance, il y a peu de travaux dans la littérature qui traitent le problème de portefeuille avec contraintes de cardinalité sur le nombre de titres. Pour sa résolution exacte, les mêmes approches que celles proposées pour les problèmes non linéaires en variables mixtes ont été utilisées. Bienstock [Bienstock, 1996] propose un algorithme branch-and-cut et Lee et Mitchell [Lee et al., 1997] proposent un algorithme branch-and-bound parallèle dans lequel l'évaluation en chaque nœud de l'arbre de recherche est calculée en résolvant la relaxation continue du problème par un algorithme de point intérieur. Concernant sa résolution approchée, Chang, Meade, Beasley et Sharaiha [Chang et al., 2000] proposent trois méthodes heuristiques fondées sur la recherche Tabou, les algorithmes génétiques et le recuit simulé. Jobst, Horniman, Lucas et Mitra [Jobst et al., 2001] proposent deux heuristiques fondées sur la résolution de la relaxation continue. Speranza [Speranza, 1996] s'est aussi intéressé à la résolution approchée du problème, mais en utilisant comme mesure du risque, la déviation absolue de l'espérance.

Pour la construction de la frontière d'efficience, il faut résoudre $(QP2)$ pour différentes valeurs de ρ. Il est peu probable de pouvoir construire la frontière point par point en utilisant une méthode exacte car cela nécessiterait sans doute un temps considérable. Seules les heuristiques présentées dans [Chang et al., 2000] et [Jobst et al., 2001] ont été developpées pour construire la frontière d'efficience du problème de portefeuille avec une contrainte sur le nombre de titres à sélectionner. Nous proposons, dans les deux paragraphes suivants, deux approches pour tracer cette frontière. Une première approche exacte qui n'est efficace que pour des petites instances du problème et une approche heuristique permettant de tracer une frontière très proche de la frontière exacte en un temps raisonnable.

2.3.2 Approche exacte

Nous avons tout d'abord appliqué une méthode exacte pour la construction de la frontière d'efficience. Cette méthode consiste à résoudre de façon exacte le problème $(QP2)$ pour différentes valeurs de ρ. Chaque problème $(QP2)$ est résolu par une procédure arborescente où l'évaluation en chaque nœud de l'arbre de recherche est obtenue par la valeur de l'optimum de la relaxation continue qui est un problème d'optimisation quadratique convexe. Afin d'avoir une bonne solution initiale (bonne borne supérieure), nous avons suivi la démarche suivante pour la construction de la frontière : on commence par la résolution du problème $(QP2)$ avec la valeur du revenu la plus forte possible puis on diminue progressivement ce revenu jusqu'à sa valeur la plus basse possible. En suivant cette procédure, on considère comme borne supérieure à l'étape k, la valeur de la solution optimale de l'étape $(k-1)$. En utilisant cette approche, nous n'avons pu tracer la frontière, en un temps de calcul raisonnable, que pour des problèmes de taille limitée (au maximum 20 titres). Pour pouvoir traiter des tailles plus grandes, nous proposons d'utiliser l'approche heuristique que nous présentons dans le paragraphe suivant.

2.3.3 Approche heuristique

Cette heuristique vise à trouver une bonne solution au problème. Elle consiste à de réduire le problème en fixant de façon heuristique un maximum de variables à zéro. On résout ensuite le problème $(QP2)$ réduit, de façon exacte, par l'algorithme branch-and-bound présenté dans le paragraphe précédent. Nous présentons ci-dessous les 4 étapes de cette approche heuristique.

Algorithm 2 Approche Heuristique

1. Résoudre la relaxation continue du problème $(QP2)$ considéré ($0 \leq y_i \leq 1$ à la place de $y_i \in \{0, 1\}$) et soit $(\tilde{x}, \tilde{y})$ la solution obtenue.
2. Soit $\varepsilon = 10^{-4}$ et $I = \{i \in \{1, ..., n\} : \tilde{x}_i \leq \varepsilon\}$; si $|I| \leq (n-K)$ alors on fixe définitivement à zéro toutes les variables x_i, $i \in I$ sinon on fixe définitivement à zéro les $(n-K)$ variables x_i ayant la plus petite valeur $\tilde{x}_i$.
3. Simplifier le problème $(QP2)$ en tenant compte des fixations effectuées au point 2.
4. Si le nombre de variables du nouveau problème est égal à K, alors on obtient une instance du problème $(QP1')$ que l'on résout par un algorithme de programmation quadratique convexe ; Fin. Sinon (le nombre de variables du problème simplifié est alors supérieur à K), on résout cette nouvelle instance du problème $(QP2)$ par une procédure branch-and-bound où l'évaluation en chaque nœud est donnée par l'optimum de la relaxation continue et où le nombre de nœuds explorés dans l'arbre de recherche est inférieur à une certaine valeur fixée à l'avance.

Pour construire la frontière d'efficience, nous avons suivi la même démarche que pour l'approche exacte en considérant les problèmes $(QP2)$ dans l'ordre décroissant du paramètre ρ.

Dans le paragraphe suivant, nous présentons les résultats numériques obtenus par cette heuristique et les comparons avec les résultats obtenus par d'autres heuristiques développées dans la littérature.

2.3.4 Résultats numériques et comparaisons

Nous avons implanté en langage C et sur un PC (Pentium II à 300MHz), d'une part, une procédure branch-and-bound permettant de déterminer les points exacts de la frontière d'efficience et, d'autre part, l'heuristique présentée dans la section précédente pour déterminer cette frontière de façon approchée. La résolution des différents programmes quadratiques convexes, $(QP1^{'})$ et la relaxation continue de $(QP2)$, a été effectuée en utilisant un outil standard de programmation mathématique : CPLEX 6.0 [Cplex, 1998]. Les expérimentations ont été réalisées sur les mêmes instances que celles qui sont considérées dans [Jobst et al., 2001] et [Chang et al., 2000]. Ces 5 instances concernent les données des cinq marchés mondiaux, Hang Seng (Hong Kong), Dax (Allemagne), FTSE (Royaume Uni), S&P (USA) et Nikkei (Japon). Pour chacune de ces instances, le nombre de titres disponibles est respectivement de 31, 85, 89, 98 et 225 et le nombre de titres à retenir pour constituer le portefeuille est égal à 10, i.e. K=10. Nous nous sommes procurés ces instances à partir de la librairie 'OR-Library'[1]. Pour chaque instance, nous avons considéré 500 valeurs de ρ.

La table 2.3 présente les résultats obtenus sur les cinq instances citées précédemment pour le calcul de 500 points exacts de la frontière d'efficience du problème avec la contrainte de cardinalité. À titre de comparaison, nous présentons, dans la table 2.4, les résultats obtenus pour la résolution exacte du problème classique de portefeuille $(QP1)$ sur ces 5 instances. Dans ces deux tables, nous indiquons pour chaque ligne, le nom de l'instance (Nom de l'instance), le nombre de titres disponibles (Nombre de titres), le nombre de points résolus (Nombre de points résolus) sur les 500 considérés dans la limite de temps fixée à 100 heures et enfin le temps total (Temps total) nécessaire au calcul de tous ces points.

[1] `http//mscmga.ms.ic.ac.uk/jeb/orlib/portfolio.html`

Nom de l'instance	**Nombre de titres**	**Nombre de points résolus**	**Temps total**
Hang Seng	31	500	565.43 secondes
DAX	85	500	$\simeq$43h
FTSE	89	450	$\simeq$100h
S&P	98	450	$\simeq$100h
Nikkei	225	500	$\simeq$100h

Tab. 2.3 – Résultats de calcul concernant la construction exacte de la FECC pour les 5 instances considérées dans le cas du problème $(QP2)$

Nom de l'instance	**Nombre de titres**	**Nombre de points résolus**	**Temps total (s)**
Hang Seng	31	500	8.62
DAX	85	500	85.5
FTSE	89	500	104.3
S&P	98	500	120.9
Nikkei	225	500	1229.61

Tab. 2.4 – Résultats de calcul concernant la construction de la FEC pour les 5 instances considérées dans le cas du problème $(QP1)$

La table 2.3 montre qu'il n'a pas été possible de déterminer les 500 points exacts de la frontière d'efficacité pour les instances FTSE et S&P. En revanche, ces 500 points ont pu être calculés pour les instances Hang Seng et Nikkei. Cependant, le calcul de ces points, dans le cas de l'instance Nikkei, a pris une centaine d'heures. Par contre, la table 2.4 montre que le temps de calcul des 500 points de la frontière d'efficacité dans le cas du problème classique $(QP1)$ a été tout à fait raisonnable pour toutes les instances.

La table 2.5 présente les résultats obtenus par l'heuristique. Pour chaque ligne, nous indiquons le nom de l'instance, le nombre de titres, le nombre de points où on trouve la solution optimale (Nombre de solutions optimales) si l'on dispose de cette dernière et le temps de résolution (Temps total). Elle montre qu'en utilisant notre heuristique, on peut résoudre des problèmes de grande taille en un temps raisonnable. Nous avons pu tracer la FECC exacte pour le problème Hang Seng (31 titres) en 42 secondes, le problème DAX (85 titres) en 1773 secondes et enfin le problème Nikkei (225 titres) en 4830 secondes.

Nom de l'instance	**Nombre de titres**	**Nombre de solutions optimales**	**Temps total**
Hang Seng	31	500	42.15
DAX	85	500	1772.8
FTSE	89	-	2167.33
S&P	98	-	2544.64
Nikkei	225	500	4829.49

TAB. 2.5 – Résultats de calcul concernant la construction de la FECC pour les 5 instances considérées par l'heuristique dans le cas du problème $(QP2)$

Afin d'analyser les résultats obtenus des expérimentations par l'heuristique, nous avons utilisé la même mesure d'erreur que dans [Jobst et al., 2001] et [Chang et al., 2000], c'est à dire la déviation des points de la FECC par rapport aux points de la FEC. La table 2.6 présente les résultats obtenus par notre heuristique pour la construction de la FECC et les compare avec ceux obtenus par l'heuristique *IntgerRestart* (un algorithme branch-and-bound) présentée dans [Jobst et al., 2001] et les heuristiques *GA* (un algorithme génétique), *TS* (un algorithme de recherche tabou), *SA* (un algorithme de récuit simulé) présentées dans [Chang et al., 2000]. Pour chaque heuristique nous indiquons le nombre de points considérés (Nb de points), l'erreur moyenne (Err_Moyenne) obtenue ainsi que le temps total de calcul donné par les auteurs (Tps_total). Les résultats montrent que notre heuristique est la plus efficace. En effet, dans la plupart des cas, l'erreur moyenne obtenue par notre heuristique est plus petite. Par ailleurs, cette heuristique est également intéressante du point de vue temps de calcul puisqu'elle est environ cinq fois plus rapide que l'heuristique proposée dans [Jobst et al., 2001].

2.4 Problème de rééquilibrage d'un portefeuille

Une question importante dans la gestion financière des portefeuilles concerne les coûts des transactions. Jusqu'à présent, nous ne les avons pas pris explicitement en compte dans les modélisations présentées. Cette partie est consacrée au problème de rééquilibrage d'un portefeuille en considèrant les coûts des transactions. Nous étudions en détail deux cas du problème : le problème de rééquilibrage avec seulement des coûts de transactions linéaires et le problème avec des coûts de transactions linéaires et fixes. Nous adaptons l'heuristique, proposée dans la deuxième partie de ce chapitre pour la résolution du problème de rééquilibrage avec des coûts linéaires et fixes. Les résultats numériques obtenus ainsi qu'une comparaison de ces derniers avec les résulats obtenus par la méthode de [Lobo et al., 2000] sont présentés.

Instance	Nb de titres	Méthode	Nb de points	Err_Moyenne	Tps_total (s)
Hang Seng	31	Integer Restart	500	1.415	57.5
		GA	1317	0.94570	
		TS	1268	0.99080	
		SA	1003	0.98920	
		Notre heuristique	500	0.3684	42.1
DAX	85	Integer Restart	500	1.399	8405.3
		GA	1270	1.95150	
		TS	1467	3.06350	
		SA	1135	2.42990	
		Notre heuristique	500	1.254	1772.8
FTSE	89	Integer Restart	500	1.141	10978.1
		GA	1482	0.87840	
		TS	1301	1.39080	
		SA	1183	1.13410	
		Notre heuristique	500	1.008	2167.3
S&P	98	Integer Restart	500	1.586	15831.9
		GA	1560	1.71570	
		TS	1587	3.16780	
		SA	1284	2.69700	
		Notre heuristique	500	1.13	2544.6
Nikkei	225	Integer Restart	500	0.618	18345.5
		GA	1823	0.6431	
		TS	1701	0.8981	
		SA	1655	0.637	
		Notre heuristique	500	0.16540	4829.5

TAB. 2.6 – Comparaison de différentes heuristiques pour la détermination de la FECC

- Integer Restart [Jobst et al., 2001]
- (GA, TS, SA) [Chang et al., 2000]

2.4.1 Définition et formulations du problème en fonction des coûts de transactions

On dispose d'un portefeuille composé de n titres. On désigne par x_i^0 la proportion du titre t_i dans le portefeuille. On cherche à constituer un portefeuille optimal à partir de ce portefeuille initial où x_i désigne la proportion du titre t_i dans ce nouveau portefeuille. Dans ce cas, le choix du nouveau portefeuille dépend non seulement de l'espérance de rendement du portefeuille et du risque associé mais aussi des coûts de transactions générés lors du passage du portefeuille initial au portefeuille cible. Ces coûts incluent par exemple les commissions, les courtages, les taxes, etc. Plusieurs types de fonctions de coûts existent. On trouve des fonctions constantes, des fonctions linéaires, des fonctions constantes par morceaux ou linéaires par morceaux (fonctions concaves), des fonctions non linéaires ou aussi des fonctions composées d'une partie linéaire et d'une partie fixe (fonctions non convexes). Dans ce travail, on se limitera aux cas des fonctions linéaires avec et sans coûts fixes.

2.4.1.1 Coûts de transactions linéaires

Dans ce cas, le coût de chaque transaction est une fonction linéaire du montant investi. Soit ϕ_i le coût associé à la transaction de chaque titre i, défini comme suit :

$$\phi_i(a_i, v_i) = \begin{cases} 0 & si\, a_i = 0\, et\, v_i = 0 \\ \rho_i^a a_i & si\, a_i \neq 0 \\ \rho_i^v v_i & si\, v_i \neq 0 \end{cases}$$

avec ρ_i^a, ρ_i^v les coûts unitaires de transaction respectivement à l'achat et à la vente du titre t_i et a_i, v_i le volume d'achat et le volume de vente du titre t_i. La fonction du coût total peut s'exprimer comme suit :

$$\phi = \sum_{i=1}^{n} \phi_i(a_i, v_i) = \sum_{i=1}^{n} \rho_i^a a_i + \sum_{i=1}^{n} \rho_i^v v_i$$

Par l'introduction de ϕ dans la fonction objectif avec le coefficient γ (un coefficient d'équilibrage), on obtient un nouveau modèle quadratique convexe $(QP3)$, défini comme suit, où x_i^0 est la proportion du titre t_i dans le portefeuille et où LB_i et UB_i, désigne respectivement, les proportions minimale et maximale du titre t_i.

$$
(QP3) \quad = \quad \left\{
\begin{array}{ll}
Min \overbrace{\sum_{i=1}^{n}\sum_{j=1}^{n} Cov_{ij}\, x_i x_j}^{Risque} + \gamma \overbrace{(\sum_{i=1}^{n} \rho_i^a a_i + \sum_{i=1}^{n} \rho_i^v v_i)}^{Co\hat{u}t\, linaire} & \\
s.c. & \\
\quad \sum_{i=1}^{n} x_i = 1 & (1) \\
\quad \sum_{i=1}^{n} E_i x_i = \rho & (2) \\
\quad \left| \begin{array}{ll} a_i \leq (UB_i - x_i^0) & i = 1, ..., n \, (3) \\ v_i \leq (x_i^0 - LB_i) & i = 1, ..., n \, (4) \\ a_i \geq x_i - x_i^0 & i = 1, ..., n \, (5) \\ v_i \geq x_i^0 - x_i & i = 1, ..., n \, (6) \end{array} \right. & \\
& \\
\quad LB_i \leq x_i \leq UB_i \quad i = 1, ..., n & \\
\quad a_i \geq 0,\, v_i \geq 0 \qquad i = 1, ..., n &
\end{array}
\right.
$$

Les contraintes (3) et (4) permettent d'imposer des bornes supérieures sur les proportions à acheter ou à vendre pour chaque titre compte tenu de la proportion maximale du titre t_i acceptée dans le portefeuille. Les contraintes (5) et (6) permettent de vérifier que la proportion achetée ou vendue d'un titre est égale à l'écart absolu entre la proportion initiale et la nouvelle proportion du titre. Similairement au cas classique de sélection d'un portefeuille initial, le problème de rééquilibrage de portefeuille, avec des coûts de transactions linéaires, est un problème quadratique convexe pour lequel il existe de multiples outils et méthodes permettant de le résoudre d'une manière efficace. Cependant en pratique, le coût d'une transaction n'est pas une fonction convexe du montant investi. Il s'ajoute, le plus souvent, un coût fixe à la partie proportionnelle. La prise en compte de la partie fixe rend le problème NP-difficile. Nous traitons ce problème dans le paragraphe suivant.

2.4.1.2 Coûts de transactions linéaires et fixes

Soit ρ_i^F le coût de transaction fixe associé à chaque titre t_i, la fonction $\phi_i(a_i,\, v_i)$ se définit alors comme suit :

$$\phi_i(a_i, v_i) = \begin{cases} 0 & si\, a_i = 0\, et\, v_i = 0 \\ \rho_i^F + \rho_i^a a_i & si\, a_i \neq 0 \\ \rho_i^F + \rho_i^v v_i & si\, v_i \neq 0 \end{cases}$$

La fonction du coût total devient : $\phi = \sum_{i=1}^{n} \rho_i^F y_i + \sum_{i=1}^{n} \rho_i^a a_i + \sum_{i=1}^{n} \rho_i^v v_i$

avec $y_i = \begin{cases} 0 & si\, x_i = x_i^0 \\ 1 & si\, x_i \neq x_i^0 \end{cases}$

Le problème de rééquilibrage de portefeuille avec un coût de transaction linéaire et un coût fixe peut se modéliser de la façon suivante :

$$(QP4) = \begin{cases} Min & \overbrace{\sum_{i=1}^{n}\sum_{j=1}^{n} Cov_{ij}\, x_i x_j}^{Risque} + \gamma\,(\overbrace{\sum_{i=1}^{n} \rho_i^F y_i}^{coût\, fixe} + \overbrace{\sum_{i=1}^{n} \rho_i^a a_i + \sum_{i=1}^{n} \rho_i^v v_i}^{coût\, lineaire}) \\ s.c. & \\ & \sum_{i=1}^{n} x_i = 1 \qquad\qquad\qquad\qquad (1) \\ & \sum_{i=1}^{n} E_i x_i = \rho \qquad\qquad\qquad\qquad (2) \\ & \left| \begin{array}{ll} a_i \leq y_i(UB_i - x_i^0) & i = 1, ..., n \qquad (3) \\ v_i \leq y_i(x_i^0 - LB_i) & i = 1, ..., n \qquad (4) \end{array} \right. \\ & \quad a_i \geq x_i - x_i^0 \qquad\quad i = 1, ..., n \qquad (5) \\ & \quad v_i \geq x_i^0 - x_i \qquad\quad i = 1, ..., n \qquad (6) \\ & LB_i y_i \leq x_i \leq UB_i y_i \quad i = 1, ..., n \\ & y_i \in \{0, 1\}^n \qquad\qquad i = 1, ..., n \\ & a_i \geq 0,\, v_i \geq 0 \qquad\quad i = 1, ..., n \end{cases}$$

Il est clair que cette modélisation est quadratique non convexe. Il n'existe pas de méthodes permettant de trouver la solution optimale sans énumération. Peu de travaux se sont intéressés à la résolution exacte du problème de rééquilibrage avec des coûts de transactions non convexes. Par contre, de nombreuses heuristiques ont été proposées pour résoudre le problème de façon approchée. Le premier qui a considéré le problème est Perold en 1984 [Perold, 1984]. Il a proposé d'approximer la fonction coût non convexe par une fonction linéaire par morceaux. Récemment, d'autres heuristiques ont été proposées. Konno et Wijayanayake [Konno et al., 2001] ont considéré le problème en utilisant le modèle MAD (Mean-absolute deviation) avec des coûts de transactions concaves. Ils proposent de remplacer la fonction coût concave par une fonction linéaire et de résoudre le problème par une méthode Branch-and-Bound. Une heuristique basée sur la relaxation SDP du problème a été proposée dans le [Caniou et al., 2000]. Lobo, Fazel et Boyd [Lobo et al., 2000] ont considéré aussi le problème de rééquilibrage avec des coûts linéaires et fixes et ont proposé une méthode itérative pour trouver une bonne solution au problème. Dans tous ces travaux, on retrouve la même idée qui consiste à approximer la fonction coût non convexe par une fonction convexe. Pour résoudre ce problème, nous avons procédé différemment, nous avons developpé une heuristique fondée sur une réduction du problème en utilisant la relaxation continue de ce dernier. Les deux paragraphes suivants décrivent cette heuristique ainsi que celle qui est présentée dans [Lobo et al., 2000]. Nous donnons des résultats numériques permettant de comparer ces deux méthodes.

2.4.2 Heuristique de Lobo, Fazel et Boyd

L'idée générale proposée dans [Lobo et al., 2000] est de remplacer la fonction des coûts de transactions par une fonction approchée convexe. On se ramène ainsi à la résolution d'un modèle quadratique convexe. Le processus itératif permettant de trouver une bonne borne supérieure commence par la résolution du problème pour une première fonction coût relaxée. Ensuite, à chaque itération, il y a une modification de ces coûts permettant d'annuler les éléments de très petits poids.

Le processus débute par la résolution du problème avec la fonction $\hat{\phi}_{1i}(a_i, v_i)$, enveloppe convexe de la fonction ϕ_i dans l'intervalle $[LB_i, UB_i]$:

$$\hat{\phi}_{1i}(a_i, v_i) = \begin{cases} 0 & si\, a_i = 0\, et\, v_i = 0 \\ \left(\dfrac{\rho_i^F}{(UB_i - x_i^0)} + \rho_i^a\right) a_i & si\, a_i \neq 0 \\ \left(\dfrac{\rho_i^F}{(x_i^0 - LB_i)} + \rho_i^v\right) v_i & si\, v_i \neq 0 \end{cases}$$

L'utilisation de $\hat{\phi}_{1i}(a_i, v_i)$ à la place de $\phi_i(a_i, v_i)$ permet de donner une solution sous-optimale qui sera utilisée pour déterminer la fonction coût de la prochaine itération. Les principales étapes du processus se résument comme suit :

Algorithm 3 Heuristique de [Lobo et al., 2000]

1. $k \leftarrow 1$

   Résoudre le problème $(QP3)$ avec la fonction coût $\hat{\phi}_i(a_i, v_i)$ définie ci-dessus.

   Soit $\left(x^1,\, v^1,\, a^1\right)$, la solution trouvée du problème.

2. $k \leftarrow k + 1$

$$\hat{\phi}_{ki}(a_i, v_i) = \begin{cases} 0 & si\, a_i = 0\, et\, v_i = 0 \\ \left(\dfrac{\rho_i^F}{a_i^{k-1}} + \rho_i^a\right) a_i & si\, a_i \neq 0 \\ \left(\dfrac{\rho_i^F}{v_i^{k-1}} + \rho_i^v\right) v_i & si\, v_i \neq 0 \end{cases}$$

   Résoudre $(QP3)$ avec la fonction coût $\hat{\phi}_{ki}(a_i, v_i)$.

   Soit $\left(x^k,\, v^k,\, a^k\right)$ la solution trouvée.

3. si les portefeuilles $\left(x^{k-1},\, v^{k-1},\, a^{k-1}\right)$ et $\left(x^k,\, v^k,\, a^k\right)$ trouvés dans les deux dernières itérations sont approximativement égaux, retourner $(x^*,\, v^*,\, a^*) = \left(x^k,\, v^k,\, a^k\right)$ et stop. Sinon aller à l'étape 2.

2.4.3 Heuristique proposée

Cette heuristique permet de trouver une bonne solution du problème. Elle est fondée sur la même idée que l'heuristique proposée dans la deuxième partie de ce chapitre pour le problème de sélection de portefeuille avec une contrainte sur le nombre de titres à sélectionner. Elle a pour but de réduire la taille du problème en fixant de façon heuristique un maximum de variables à zéro. On résout ensuite le problème $(QP4)$ réduit par un algorithme branch-and-bound tronqué. Nous présentons ci-après les quatre étapes de cette approche heuristique.

Algorithm 4 Heuristique proposée

1. Résoudre la relaxation continue du problème $(QP4)$ considéré ($0 \leq y_i \leq 1$ à la place de $y_i \in \{0, 1\}$) et soit $(\tilde{x}, \tilde{y}, \tilde{v}, \tilde{a})$ la solution obtenue ;
2. Soit $\varepsilon = 10^{-4}$ et $I = \{i \in \{1, ..., n\} : \tilde{x}_i \leq \varepsilon\}$; on fixe définitivement à zéro toutes les variables x_i, $i \in I$.
3. Simplifier le problème $(QP4)$ en tenant compte des fixations effectuées au point 2.
4. Résoudre cette nouvelle instance du problème $(QP4)$ par une procédure branch-and-bound où l'évaluation en chaque nœud est donnée par l'optimum de la relaxation continue et où le nombre de nœuds explorés dans l'arbre de recherche est inférieur à une certaine valeur fixée à l'avance.

2.4.4 Résultats Numériques et comparaison

L'heuristique proposée ainsi que l'heuristique de [Lobo et al., 2000] ont été implantées en langage C sur un PC de type pentium II à 300 MHz. Pour la résolution de la relaxation continue ainsi que pour l'évaluation en chaque nœud du branch-and-bound, nous avons utilisé l'outil CPLEX 6.0 [Cplex, 1998]. Les tests numériques ont été effectués sur les mêmes instances que celles qui sont utilisées dans le paragraphe 2.3.4. Les coûts unitaires d'achat et de vente ont été fixés à 1, les coûts fixes à 10 et le portefeuille initial est composé de n titres avec des proportions égales : $x_i^0 = 1/n \, \forall i \in \{1, ..., n\}$. Pour chaque instance, nous avons considéré 100 valeurs de ρ.

Le tableau 2.7 concerne la résolution exacte du problème, avec et sans coûts fixes, obtenue par une procédure branch-and-bound où l'évaluation en chaque nœud est donnée par l'optimum de la relaxation continue. Pour chaque problème, nous indiquons le temps total de résolution (temps total), c'est-à-dire le temps nécessaire pour résoudre les 100 instances, ainsi que le nombre d'instances que nous avons pu résoudre (Nb_instances résolues). Nous avons choisi de limiter le temps de résolution d'une instance à 1 heure.

Le tableau 2.7 montre que la résolution du problème de rééquilibrage devient beaucoup plus difficile avec la présence de coûts fixes. Nous n'avons pu résoudre que le problème de taille

Problème	Nb_titre	Problème sans coûts fixes		Problème avec coûts fixes	
		Temps total	Nb_instances résolues	Temps total	Nb_instances résolues
Hang Seng	31	5.17 s	100	$\simeq$18h	100
DAX	85	43.29 s	100	100h	0
S&P	98	73.56 s	100	100h	0
Nikkei	225	526.65 s	100	100h	0

TAB. 2.7 – Résolution exacte du problème de rééquilibrage

31, de plus en un temps assez important.

Le tableau 2.8 compare notre heuristique et celle de [Lobo et al., 2000]. Pour chacune, on indique le temps total (Temps total) pour la résolution des 100 instances ainsi que l'écart relatif en moyenne ($\frac{UB-OPT}{OPT}$) pour les instances dont nous connaissons la solution exacte. Nous précisons aussi l'écart relatif entre notre heuristique et celle de [Lobo et al., 2000].

Le tableau 2.8 montre l'efficacité de notre heuristique. En comparaison avec la résolution exacte, il est clair que notre approche heuristique est beaucoup plus rapide. Par exemple, le temps total mis par l'heuristique proposée pour la résolution de 100 instances du problème Hang Seng est 47 secondes alors qu'il est égal à 18 heures pour la résolution exacte. L'écart relatif est très petit et est égal à 0.43%. En comparaison avec l'heuristique de [Lobo et al., 2000], on constate une amélioration en temps de résolution de plus que 50% pour trois instances sur quatre. Une amélioration en qualité de solutions est également constatée, l'ecart relatif entre notre heuristique et celle de [Lobo et al., 2000] est en moyenne de 2.5%.

Problème	Nb_titre	Heuristique de Lobo et al.		Notre heuristique		
		Temps total(s)	Ecart rélatif (ub1-opt)/opt	Temps total (s)	Ecart rélatif (ub2-opt)/opt	Ecart rélatif (ub1-ub2)/ub2
Hang Seng	31	37.40	2.1%	47.30	0.43%	1.65%
DAX	85	254.89	-	83.76	-	4.36%
S&P	98	378.60	-	145.16	-	3.02%
Nikkei	225	3419.40	-	1476.64	-	0.50%

TAB. 2.8 – Comparaison de notre heuristique avec celle de [Lobo et al., 2000]
opt : valeur de la solution optimale obtenue par la résolution exacte du problème.
ub1, ub2 : valeurs de solutions approchées trouvées respectivement par l'heuristique de [Lobo et al., 2000] et par notre heuristique.

2.5 Conclusion

Nous avons étudié, dans ce chapitre, le problème de la gestion optimale de portefeuilles en utilisant le modèle de Markowitz [Markowitz, 1952]. Nous avons considéré deux problèmes : la construction de la frontière d'efficience d'un portefeuille avec une contrainte sur le nombre de titres à sélectionner, et le rééquilibrage d'un portefeuille avec des coûts de transactions fixes et linéaires. Nous avons formulé ces problèmes comme des programmes quadratiques en variables mixtes qui sont des programmes non convexes. Nous avons proposé une méthode branch-and-bound pour les résoudre de façon exacte. L'évaluation en chaque nœud de l'arbre de recherche est donnée par la relaxation continue du programme qui est un programme quadratique convexe. Pour la résolution des instances de grande taille, nous avons proposé des méthodes heuristiques, fondées sur la relaxation continue du modèle, couplée avec une procédure branch-and-bound tronquée. Les résultats des expérimentations numériques réalisées sur les données de cinq marchés mondiaux montrent l'efficacité de cette approche heuristique.

Bibliographie

[Bienstock, 1996] D. Bienstock, 'Computational stydy of a family of mixed-integer quadratic programming problems', Mathematics Programming, Vol. 74, p. 121-140 (1996).

[Billionnet et al., 2002a] A. Billionnet et K. Djebali, 'Optimisation de portefeuille avec un nombre de titres fixé', 4^e congrès de la société française de recherche opérationnelle et d'ide à la décision ROADEF 2002, Télécom Paris (2002).

[Billionnet et al., 2002b] A. Billionnet et K. Djebali, 'Problème de rééquilibrage d'un portefeuille avec des coûts de transactions linéaires et fixes' CIRO'02, IIIème Conférence Internationale en Recherche Opérationnelle, Marrakech, Maroc (2002).

[Billionnet et al., 2003] A. Billionnet et K. Djebali, 'Heuristic for finding the efficient frontier in cardinality constrained portfolio optimisation', Rapport de recherche Nř 514, CEDRIC, Cnam, Paris (2003).

[Caniou et al., 2000] A. Caniou, J. Malick, M.L. Overton, F. Oustry, C. Lemarechal, 'SDP relxation of portfolio selection with transaction costs problem', Rapport de recherche, Courant Institute of Mathematical Sciences of New York University & Inria Montbonnot (2000).

[Chang et al., 2000] T.J. Chang, N. Meade, J.E. Beasley, Y.M. Sharaiha, 'Heuristics for cardinality constrained portfolio optimization', Computers&Operations Research, Vol. 27, p. 1271-1302 (2000).

[Cplex 6.0, 1998] Ilpg, Inc., Cplex Division, 'Using the Cplex callable library', Version 6.0 (1998).

[Jobst et al., 2001] N.J. Jobst, M.D. Horniman, C.A. Lucas, G. Mitra, 'Computational aspects of alternative portfolio selection models in the presence of discrete asset choice constraints', Quantitative Finance, Vol. 1, p. 1-13 (2001).

[Konno et al., 1991] H. Konno, H. Yamazaki, 'Mean-Absolute Deviation Portfolio Optimization Model and Its applications to Tokyo Stock Market', Management Science, Vol. 37, p. 519-531 (1991).

[Konno et al., 2001] H. Konno, A. Wijayanayake, ' Portfolio optimization problem under concave transaction costs and minimal transaction unit constraints', Mathematical Programming, Ser B, Vol. 89, p. 233-250 (2001).

[Lee et al., 1997] E.K. Lee, J.E. Mitchell, 'Computational experience of an interior-point SQP algorithm in a parallel branch-and-bound framework', Proceedings of High Performance Optimization Techniques, Springer Verlag, (1997).

[Lobo et al., 2000] M.S. Lobo, M. Fazel and S. Boyd, 'Portfolio Optimization with linear and fixed transaction costs and bounds on risk', Mimeo, Information Systems Laboratory, Standford University (2000).

[Markowitz, 1952] H. Markowitz, 'Portfolio selection', The Journal of Finance, Vol. 7, p. 77-91 (1952).

[Perold, 1984] A.F. Perold, 'Large scale portfolio optimization', Management Science, Vol. 30, p. 1143-1160 (1984).

[Rosenberg, 1974] B. Rosenberg, 'Extra-market components of covariance in security returns', Journal of Financial Quantitative Analysis, Vol. 9, p. 263-273 (1974).

[Sharpe, 1963] W.F. Sharpe, 'A simplified model for portfolio analysis', Management Science, Vol. 9, p. 277-293 (1963).

[Speranza, 1996] M.G. Speranza, 'A heuristic algorithm for a portfolio optimization model applied to the Milan stock market', Computers&Operations Research, Vol. 23, p. 433-441 (1996).

[Young, 1998] M.R. Young, 'A Minimax Portfolio Selection Rule with Linear Programming Solution', Management Science, Vol. 44, p. 673-683 (1998).

Chapitre 3

Maximisation de la somme de plusieurs ratios hyperboliques en 0-1

3.1 Introduction

Les programmes fractionnaires linéaires ou non linéaires, en continu, en nombres entiers ou en variables 0-1 sont utilisés dans plusieurs domaines tels que les bases de données, l'optimisation combinatoire, la programmation stochastique et l'économie [Stancu-Minasian, 1997]. De nombreuses applications des programmes fractionnaires ont été décrites dans la littérature [Schaible, 1981], [Schaible et al., 1983], [Hansen et al., 1991], [Falk et al., 1992], [Chen et al., 2000], [Freund et al., 2001].

Dans ce chapitre, nous considérons un problème d'optimisation combinatoire fractionnaire : la maximisation de la somme de plusieurs ratios hyperboliques. La première partie de ce chapitre est consacrée à une présentation générale des travaux développés dans le cas simple d'optimisation d'un seul ratio. Nous passons en revue les différentes méthodes de résolution de ce problème. Dans la deuxième partie, nous étudions le problème de l'optimisation de la somme de plusieurs ratios hyperboliques. Nous présentons trois formulations de ce problème par des programmes linéaires en variables mixtes et proposons trois stratégies de résolution. Tout d'abord, une résolution directe en utilisant un outil général de programmation linéaire en variables mixtes. Ensuite, une résolution heuristique également exploitée pour améliorer la première stratégie en fournissant au solveur une bonne solution admissible initiale. Enfin, une résolution par un algorithme branch-and-bound spécifique, fondé sur une réécriture plus précise des programmes en chaque nœud de l'arbre de recherche, cette réécriture consistant à ajuster les coefficients des programmes. Nous présentons des résultats expérimentaux permettant de comparer les différentes stratégies.

3.2 Optimisation fractionnaire combinatoire

Étant donné X un ensemble discret, f et g deux fonctions définies sur X, le problème d'optimisation combinatoire fractionnaire consiste à déterminer un élément x^* de X optimisant le rapport $f(x)/g(x)$ pour $x \in X$. Lorsque les fonctions f et g sont linéaires, le problème est dit fractionnaire linéaire, ou encore hyperbolique. Il s'ecrit comme suit :

$$(FP) \begin{cases} Min\, ou\, Max\, \dfrac{a_0 + ax}{b_0 + bx} \\ x \in X \end{cases}$$

où a_0 et b_0 sont des réels, a et b sont des vecteurs de $\mathbb{R}^n$. On suppose de plus que $(b_0 + bx) > 0$ pour tout $x \in X$.

De nombreuses applications de l'optimisation combinatoire fractionnaire existent dans la littérature. Citons, par exemple, le problème de recherche d'un arbre couvrant de longueur moyenne minimale [Chandrasekaran, 1977], le problème de recherche d'un cycle profit/temps maximum (maximum profit-to-time cycle en anglais)[Radzik, 1998], le problème de recherche d'une coupe de poids moyen pondéré maximum (maximum mean-weight cut en anglais) [Radzik, 1998], le problème du sac à dos fractionnaire [Nagih et al., 1995], [Billionnet et al., 2002], le problème d'optimisation de requêtes en recherche documentaire [Hansen et al., 1991] et le problème de génération de colonnes [Hansen et al., 1987].

Du fait de la particularité de ce problème, une large gamme de méthodes de résolution spécifiques ont éte envisagées (voir, par exemple, [Nagih et al., 1999] pour un état de l'art détaillé sur l'optimsation d'un seul ratio hyperbolique). Ces méthodes se sont avérées plus efficaces que l'application directe de méthodes générales de programmation non linéaire. Nous retrouvons, dans la littérature, trois principales approches : la résolution directe, la résolution par paramétrisation, et la résolution par transformation en un programme équivalent. Nous présentons ci-après ces trois approches.

3.2.1 La résolution directe

Cette approche a été proposée pour résoudre les programmes hyperboliques en 0-1 sans contraintes, définis comme suit : $P = \left\{ max \left(a_0 + \sum_{j=1}^n a_j x_j \right) \Big/ \left(b_0 + \sum_{j=1}^n b_j x_j \right) \,:\, x \in \{0,1\}^n \right\}$. Dans le cas général, ce problème est NP-difficile [Hansen et al., 1991], cependant, lorsque tous les coefficients sont positifs, il existe des algorithmes exacts de complexité quadratique voire linéaire. Ces algorithmes, proposés par Hammer et Rudeanu [Hammer et al., 1968], Robillard [Robbilard, 1971] et Hansen, Poggi et Robeiro [Hansen et al., 1991] s'appuient sur deux propriétés principales.

Propriété *1 : Lorsque tous les coefficients sont positifs, le rapport de la somme des coefficients de deux variables est compris entre le minimum et le maximum des rapports de chaque variable.*

$$min \left\{ \frac{a_j}{b_j}, \frac{a_k}{b_k} \right\} \leq \frac{a_j + a_k}{b_j + b_k} \leq max \left\{ \frac{a_j}{b_j}, \frac{a_k}{b_k} \right\} \;\; \forall\, j,\, k \in \{1, ..., n\}$$

Propriété *2 : Lorsque tous les coefficients sont positifs, on peut simplifier le problème* (P) *en fixant définitivement à zéro toutes les variables ayant un rapport* (a_j/b_j) *inférieur à* (a_0/b_0) : $x_j^* = 0$ *pour tout* $j \in \{1, ..., n\}$ *tel que* $\dfrac{a_j}{b_j}$ *est inférieur ou égal à la valeur d'une solution admissible.*

Nous détaillons ci-après différents algorithmes de résolution directe.

3.2.1.1 Algorithme de Hammer et Rudeanu

Hammer et Rudeanu proposent dans [Hammer et al., 1968] un algorithme (cf. Alg. 5) de complexité quadratique. L'idée générale de l'algorithme se base sur la propriété suivante :

Propriété *3 :* $x_k^* = 1$ *pour tout* $k \in \{1, ..., n\}$ *tel que* $\dfrac{a_k}{b_k} = max \left\{ \dfrac{a_j}{b_j} : j \in \{1, ..., n\} \right\}$

À chaque itération de l'algorithme, la variable x_k ayant le plus grand rapport a_j/b_j est fixée définitivement à 1 et les variables x_k ayant un rapport inférieur ou égal à la valeur de la meilleure solution admissible sont fixées définitivement à 0.

Algorithm 5 Algorithme de Hammer et Rudeanu

Début

{initialisation}

$J \leftarrow \{1, ..., n\}$

Répéter

déterminer k tel que $\dfrac{a_k}{b_k} = max \left\{ \dfrac{a_j}{b_j} | j \in J \right\}$;

$x_k^* \leftarrow 1$; $a_0 \leftarrow a_0 + a_k$; $b_0 \leftarrow b_0 + b_k$;

$J_0 \leftarrow \left\{ j \in J \left| \dfrac{a_j}{b_j} \leq \dfrac{a_0}{b_0} \right. \right\}$; $J \leftarrow J - J_0$- $\{k\}$;

fixer x_j^* à 0 pour tout $j \in J_0$

Jusqu'à *fixer toutes variables* $(J = \emptyset)$

Fin

3.2.1.2 Algorithme de Robillard

Cet algorithme (cf. Alg. 6) est également de complexité quadratique. Il repose sur la propriété suivante.

***Propriété** 4 : En supposant que* $\frac{a_1}{b_1} \leq ... \leq \frac{a_n}{b_n}$*, soit* k *un entier dans* $\{1, ..., n\}$ *tel que :* $\frac{a_0 + \sum_{j=k}^{n} a_j}{b_0 + \sum_{j=k}^{n} b_j} < \frac{a_i}{b_i}$ *pour tout* $i \geq k$ *et* $\frac{a_0 + \sum_{j=k}^{n} a_j}{b_0 + \sum_{j=k}^{n} b_j} \geq \frac{a_i}{b_i}$ *pour tout* $i < k$ *alors toute solution optimale* x^* *est définie par :* $x_j^* = 1$ si $j \geq k$ et $x_j^* = 0$ si $j < k$

Algorithm 6 Algorithme de Robillard

Début

{initialisation}

$J_0 \leftarrow \emptyset$; $J_1 \leftarrow \emptyset$;

Pour j *de 1 à* n ***faire***

si $\frac{a_j}{b_j} > \frac{a_0}{b_0}$ **alors** $J_1 \leftarrow J_1 \cup \{j\}$; $a_0 \leftarrow a_0 + a_j$; $b_0 \leftarrow b_0 + b_j$;

répéter

$J_0 \leftarrow \left\{ i \in J_1 : \frac{a_i}{b_i} \leq \frac{a_0}{b_0} \right\}$; $J_1 \leftarrow J_1 - J_0$;

$a_0 \leftarrow a_0 - \sum_{i \in J_0} a_i$; $b_0 \leftarrow b_0 - \sum_{i \in J_0} b_i$;

jusqu'à $J_0 = \emptyset$;

finsi

finpour

Pour j *de 1 à* n ***faire***

si $j \in J_1$ ***alors*** $x_j^* \leftarrow 1$ ***sinon*** $x_j^* \leftarrow 0$ ***finsi***

finpour

Fin

3.2.1.3 Algorithme de Hansen-Poggi-Ribeiro

Cet algorithme (cf. Alg. 7) est de complexité linéaire. Il procède par recherche dichotomique et se base sur la propriété suivante.

Propriété *5 : Soit λ^* la valeur optimale du problème, alors toute solution optimale vérifie :*

$x_j^* = 1$ *si* $a_j/b_j > \lambda^*$

$x_j^* = 0$ *si* $a_j/b_j < \lambda^*$

$x_j^* = 0\,ou\,1$ *si* $a_j/b_j = \lambda^*$

À chaque itération de l'algorithme, on essaye de trouver λ^* la valeur optimale du problème. Cela se fait en utilisant la médiane des variables non encore fixées et en appliquant la propriété 5. On s'arrête si toutes les variables sont fixées.

Algorithm 7 Algorithme de Hansen-Poggi-Ribeiro

Début

$\overline{a} \leftarrow a_0\,;\ \overline{b} \leftarrow b_0\,;\ J \leftarrow \{1,\ ...,\ n\}\,;\ fin \leftarrow faux\,;$

répéter

rechercher k tel que $a_k/b_k = Mdiane\,de\,la\,liste\ \{a_j/b_j,\ j \in J\}\,;$

$J_1 \leftarrow \{j \in J\,|(a_j/b_j) < (a_k/b_k)\}\,;$

$J_2 \leftarrow \{j \in J\,|(a_j/b_j) = (a_k/b_k)\}\,;$

$J_3 \leftarrow \{j \in J\,|(a_j/b_j) > (a_k/b_k)\}\,;$

$\overline{a} \leftarrow a_0 + \sum_{j\in J_2\cup J_3} a_j\,;\ \overline{b} \leftarrow b_0 + \sum_{j\in J_2\cup J_3} b_j\,;\ \lambda \leftarrow \overline{a}/\overline{b}\,;$

si $\lambda > a_k/b_k$ ***alors***

$x_j^* \leftarrow 0\,\forall\, j \in J_1 \cup J_2\,;\ J \leftarrow J_3\,;$

sinon

si $J_1 \neq \emptyset$ ***alors***

déterminer le plus grand élément $\dfrac{a_k}{b_k}$ de la liste $\left\{\dfrac{a_j}{b_j}, j \in J_1\right\}\,;$

si $\lambda < \dfrac{a_k}{b_k}$ ***alors***

$x_j^* \leftarrow 1\,\forall\, j \in J_2 \cup J_3\,;\ J \leftarrow J_1\,;\ a_0 \leftarrow \overline{a}\,;\ b_0 \leftarrow \overline{b}\,;$

sinon

$x_j^* \leftarrow 0\,\forall j \in J_1\,;\ x_j^* \leftarrow 1\,\forall\, j \in J_2 \cup J_3\,;\ fin \leftarrow vrai\,;$

finsi

finsi

finsi

Jusqu'à *fin*

$\lambda^* \leftarrow \lambda$

Fin

Une amélioration de cet algorithme a été proposée par Nagih et Plateau [Nagih et al., 2000].

3.2.2 La résolution par paramétrisation

Cette approche a été utilisée pour différents programmes fractionnaires, linéaires et non linéaires, en variables continues comme en variables discrètes. Proposée initialement en 1956 par Isbell et Marlow [Isbell et al. 1956] pour la résolution des programmes fractionnaires linéaires, cette approche a été généralisée en 1967 par Dinkelbach [Dinkelbach, 1967] au cas non linéaire. L'idée est de simplifier l'objectif du programme mathématique. Pour cela, on introduit un paramètre permettant de transformer un programme hyperbolique en un programme paramétré linéaire, ou un programme fractionnaire quadratique en un programme paramétré quadratique. Ainsi le programme obtenu peut être résolu “paramétriquement” : une séquence de résolutions de tels programmes à objectifs simplifiés engendre une suite de solutions convergeant vers une solution optimale du programme fractionnaire initial. Différents algorithmes incluant des algorithmes de résolution itératifs et dichotomiques ont été proposés autour de cette approche. Les propriétés fondamentales du programme paramétré et les principaux algorithmes de résolution sont présentés ci-après.

3.2.2.1 Description et propriétés du problème paramétré

Soit le programme fractionnaire : $(P) = \{max\, f(x)/g(x) \, : \, x \in X\}$, dont le programme paramétré associé est défini comme suit : $(P(\lambda)) = \{max\, f(x) - \lambda g(x) \, : \, x \in X\}$. Ce dernier consiste à simplifier l'objectif en combinant le numérateur et le dénominateur à l'aide d'un paramètre $\lambda \in \mathbb{R}$. Ce problème paramétré vérifie les propriétés décrites dans [Dinkelbach, 1967] et que nous résumons ci-après. Soient λ^* la valeur optimale de (P) et $v(\lambda)$ la valeur optimale de $(P(\lambda))$.

Propriété 6 : toute solution optimale de $P(\lambda^*)$ est une solution optimale de P.

Propriété 7 : la fonction $\lambda \rightarrow v(\lambda)$ est continue, strictement décroissante, convexe.

Propriété 8 : $v(\lambda) = 0$ si et seulement si λ est égal à λ^* .

Propriété 9 :

$$\begin{aligned} v(\lambda) = 0 &\Leftrightarrow \lambda = \lambda^* \\ v(\lambda) > 0 &\Leftrightarrow \lambda < \lambda^* \\ v(\lambda) < 0 &\Leftrightarrow \lambda > \lambda^* \end{aligned}$$

Ainsi la résolution de (P) revient à trouver la racine de l'équation non linéaire à une seule variable : $v(\lambda) = 0$.

3.2.2.2 Méthode de Newton

Cette méthode (cf. Alg 8) proposée initialement par Isbell et Marlow [Isbell et al. 1956] pour la résolution des programmes fractionnaires linéaires a été généralisée par Dinkelbach [Dinkelbach, 1967] pour le cas non linéaire. Partant d'un λ^0 vérifiant $v(\lambda^0) > 0$ ($\lambda^0 < \lambda^*$), cette méthode produit une suite de réels $(\lambda^k)_{k\in\mathbb{N}}$, convergeant vers λ^*, chaque élément λ^k de la suite étant égal à $f(x^{k-1})/g(x^{k-1})$ où x^{k-1} est une solution optimale de $(P(\lambda^{k-1}))$.

Algorithm 8 Algorithme de Newton

Début

{initialisation}

$\lambda \leftarrow \lambda^0$ $(v(\lambda^0) > 0)$;

Répéter

déterminer x_λ solution optimale de $(P(\lambda))$;

$\lambda \leftarrow f(x_\lambda)/g(x_\lambda)$;

Jusqu'à $v(\lambda) = 0$

Fin

3.2.2.3 Méthode d'interpolation linéaire

Schaible [Schaible, 1976] propose une méthode (cf. Alg. 9) basée sur la notion d'interpolation linéaire. Á l'inverse de la méthode de Newton, elle génère une suite de réels convergeant vers λ^* par valeurs supérieures. Cette suite est obtenue par les abscisses des intersections d'un ensemble de courbes (droites) avec l'axe des λ. Soient λ^l et λ^u deux réels tels que $v(\lambda^l) > 0$ et $v(\lambda^u) < 0$, et x^l et x^u les solutions optimales respectives de $P(\lambda^l)$ et $P(\lambda^u)$ et soit la droite d'equation $D(\lambda) = \dfrac{v(\lambda^l) - v(\lambda^u)}{(\lambda^l - \lambda^u)}\lambda - \dfrac{\lambda^l v(\lambda^u) - \lambda^u v(\lambda^l)}{(\lambda^l - \lambda^u)}$ joignant les points $(\lambda^l, v(\lambda^l))$ et $(\lambda^u, v(\lambda^u))$. À chaque itération de l'algorithme, on cherche la nouvelle valeur de λ obtenue par l'intersection de la droite $D(\lambda)$ avec l'axe des λ : $\lambda = \dfrac{\lambda^u v(\lambda^l) - \lambda^l v(\lambda^u)}{v(\lambda^l) - v(\lambda^u)}$. Ensuite, on calcule $v(\lambda)$, la valeur de la solution optimale de $P(\lambda)$. On s'arrête quand $v(\lambda) = 0$.

Algorithm 9 Algorithme d'interpolation linéaire

Début

{initialisation}

déterminer λ^l et λ^u tels que $\lambda^l < \lambda^* < \lambda^u$ et calculer $v(\lambda^l)$ et $v(\lambda^u)$;

Répéter

$\lambda \leftarrow \dfrac{\lambda^u v(\lambda^l) - \lambda^l v(\lambda^u)}{v(\lambda^l) - v(\lambda^u)}$;

déterminer $v(\lambda)$ valeur de la solution optimale de $(P(\lambda))$;

Jusqu'à $v(\lambda) = 0$

Fin

3.2.2.4 Méthode de recherche dichotomique

Ibaraki [Ibaraki, 1983] propose une méthode (cf. Alg. 10) basée sur la recherche dichotomique. On considère un intervalle $[\lambda^l, \lambda^u]$ contenant λ^* et on essaye de réduire cet intervalle par recherche dichotomique. On s'arrête lorsque l'ecart entre λ^l et λ^u devient trop petit. La convergence de cette méthode est linéaire.

Algorithm 10 Algorithme de recherche dichotomique

Début

{initialisation}

déterminer λ^l et λ^u tels que $\lambda^l < \lambda^* < \lambda^u$;

Répéter

$\lambda \leftarrow (\lambda^l + \lambda^u)/2$;

déterminer $v(\lambda)$ la valeur de la solution optimale de $(P(\lambda))$;

si $v(\lambda) > 0$ ***alors*** $\lambda^l \leftarrow \lambda$ ***sinon*** $\lambda^u \leftarrow \lambda$ ***finsi***

Jusqu'à $v(\lambda) = 0$

Fin

3.2.2.5 Méthode de Meggido

La méthode de Meggido (cf. Alg. 11) n'est autre qu'une recherche dichotomique. Étant donné un intervalle $[\lambda^l, \lambda^u]$ contenant λ^* et $\lambda^1 \in \left[\lambda^l, \lambda^u\right]$, si $v(\lambda^l) > 0$ ($\lambda^l < \lambda^*$ d'après la propriété 9) alors $\lambda^l \leftarrow \lambda^u$ sinon, $\lambda^u \leftarrow \lambda^l$. De plus, un résultat très important a été montré par Meggido. Il montre que si le problème paramétré $(P(\lambda))$ peut être résolu avec $O(p(n))$ comparaisons et $O(q(n))$ additions, alors le problème (P) est solvable en $(O(p(n)\,(q(n)+p(n)))$.

Algorithm 11 Algorithme de Meggido

Début

{initialisation}

déterminer λ^l et λ^u tels que $\lambda^l < \lambda^* < \lambda^u$;

répéter

$\lambda \leftarrow \lambda^l$;

déterminer $x_\lambda$solution optimale de $(P(\lambda))$ avec un algorithme polynomial ;

si $v(\lambda) > 0$ ***alors*** $\lambda^l \leftarrow \lambda$ ***sinon*** $\lambda^u \leftarrow \lambda$ ***finsi***

Jusqu'à $v(\lambda) = 0$

Fin

3.2.3 La résolution par transformation en un programme équivalent

La simplification de l'objectif peut être faite, entre autres, par changement de variables. Cependant, à l'inverse de l'approche paramétrée, ce changement de variables induit l'ajout de variables et de contraintes supplémentaires. Ainsi, la transformation en un programme linéaire équivalent permet l'application des algorithmes standards. La transformation la plus connue a été proposée par Charnes et Cooper [Charnes et al., 1962] pour des programmes fractionnaires continus. Dans le cas du ratio de deux fonctions linéaires, elle consiste à effectuer le changement de variables suivant : $y = (1/(b_0 + bx))x$ et $t = 1/(b_0 + bx)$. Le programme linéaire équivalent obtenu est comme suit :

$$(Peq)\begin{cases} max/min\ a_0t + ay \\ b_0t + by = 1 \\ t > 0,\, y/t \in X \end{cases}$$

Proposition 1 ([Charnes et al., 1962]) : si (y^*, t^*) est une solution optimale de (Peq), alors $t^* > 0$ et $x^* = y^*/t^*$ est une solution optimale de (P).

On peut adapter cette technique aux problèmes hyperboliques binaires. Dans ce cas les variables y_i ne peuvent prendre que les valeurs 0 et t. On ajoutera donc n variables binaires $(z_1, z_2, ..., z_n)$ et les contraintes suivantes : $y_i \leq t$, $y_i \leq z_i x_i$, $y_i \geq t - 1 + z_i$ $(i = 1, ..., n)$.

En dépit de ces diverses méthodes de résolution, il reste un grand nombre de problèmes fractionnaires combinatoires difficiles à résoudre. En effet, ces méthodes ne sont adéquates que pour l'optimisation d'un seul ratio hyperbolique. Parmi les problèmes difficiles, on retrouve le problème fractionnaire multi-objectifs, le problème MaxMin (ou MinMax) fractionnaire, le problème de la somme de ratios, etc. [Schaible, 1995]. Dans la suite de ce chapitre, nous étudions un de ces problèmes, celui de la somme de plusieurs ratios hyperboliques en 0-1.

3.3 Le problème de la somme de plusieurs ratios

Le problème considéré est un problème en variables 0-1 et consiste à optimiser la somme de plusieurs ratios hyperboliques. Il s'écrit comme suit :

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

où $a_i \in \mathbb{R}$, $b_i \in \mathbb{R}$, $a_{ij} \in \mathbb{R}$ et $b_{ij} \in \mathbb{R}$. Nous considérons, de plus, l'hypothèse classique $(b_i + \sum_{j=1}^{n} b_{ij} x_j) > 0$ $\forall x \in \{0, 1\}^n$, $\forall i \in \{1, ..., m\}$ et nous posons $N = \{1, ..., n\}$ et $M = \{1, ..., m\}$.

Diverses applications peuvent se modéliser sous la forme (SRH) (voir [Arora et al., 1977], [Gilmore et al., 1963], [Hansen et al., 1991]). Contrairement à la maximisation d'un seul ratio hyperbolique, peu d'études traitent ce problème. En 1975, Saipe [Saipe, 1975] s'est intéressé à ce problème avec une contrainte de cardinalité et a proposé de le résoudre par un algorithme branch-and-bound. Plus récemment, Li [Li, 1994] et Wu [Wu, 1997] ont formulé le problème par un programme linéaire en variables mixtes (PLVM) et ont proposé de résoudre ce programme par un algorithme branch-and-bound standard. Cette technique de modélisation 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. Dans ce travail, nous présentons la formulation de [Wu, 1997] ainsi que trois nouvelles formulations. Pour la résolution des différentes linéarisations considérées, nous proposons trois approches. La première consiste à utiliser un outil de programmation linéaire mixte pour résoudre le problème par un algorithme branch-and-bound standard. La deuxième consiste à le résoudre par une méthode heuristique. La solution approchée obtenue sera également exploitée pour améliorer la première approche. La troisième utilise un algorithme branch-and-bound spécifique fondé sur une réécriture plus précise des programmes en chaque nœud de l'arbre de recherche. Des résultats expérimentaux seront présentés afin de comparer les différentes linéarisations et approches de résolution.

3.3.1 Différentes linéarisations du problème

Dans cette partie, nous présentons différentes techniques de linéarisation qui transforment le problème SRH en un problème linéaire en variables mixtes. Nous exposons tout d'abord la formulation de Wu [Wu, 1997] puis nous proposons trois nouvelles formulations.

3.3.1.1 Linéarisation de Wu

Proposée par [Wu, 1997], cette linéarisation consiste à remplacer l'inverse des dénominateurs des ratios par des variables continues $y_i = 1/(b_i + \sum_{j=1}^{n} b_{ij}x_j)$ puis à linéariser le programme obtenu. On obtient ainsi, après changement de variables, le programme quadratique suivant :

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

Une façon classique de linéariser ce programme est de remplacer le produit de deux variables (une variable bivalente et une variable continue) par une seule variable tout en ajoutant des contraintes appelées contraintes de linéarisation.

Lemme 1 **([Wu, 1997]) :** Soit $x \in \{0, 1\}$, y une variable continue non-négative et K une borne supérieure de y. Le produit xy peut être remplacé par la variable z soumise à l'ensemble de contraintes suivant :

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

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

Appliquons le Lemme 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 1 où $\overline{y}_i$ est une borne supérieure de y_i.

Théorème 1 : Soit (x^*, y^*, z^*) une solution optimale de $LIN1$. Alors x^* est une solution optimale de SRH.

$$(LIN1)\begin{cases} Max & \sum_{i=1}^{m}(a_i y_i + \sum_{j=1}^{n} a_{ij} z_{ij}) & \\ s.c. & b_i y_i + \sum_{j=1}^{n} b_{ij} z_{ij} = 1 & i \in M \\ & z_{ij} \geq y_i - \bar{y}_i(1-x_j) & i \in M,\, j \in N\ (1) \\ & z_{ij} \leq y_i & i \in M,\, j \in N\ (2) \\ & z_{ij} \leq \overline{y}_i x_j & i \in M,\, j \in N\ (3) \\ & x_j \in \{0,\, 1\},\, z_{ij} \geq 0 & i \in M,\, j \in N \end{cases}$$

3.3.1.2 Une nouvelle linéarisation

L'idée de cette linéarisation est d'écrire la fonction objectif comme la somme de m variables r_i, $i \in M$ où $r_i = \left(a_i + \sum_{j=1}^{n} a_{ij} x_j\right) \bigg/ \left(b_i + \sum_{j=1}^{n} b_{ij} x_j\right)$.

Le programme SRH après changement de variables devient :

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

Ce nouveau programme est aussi non linéaire. En remplaçant le produit $r_i x_j$ par la variable z_{ij} et en appliquant le Lemme 1, nous obtenons le PLVM suivant :

$$\begin{cases} Max & \sum_{i=1}^{m} r_i \\ s.c. & a_i + \sum_{j=1}^{n} a_{ij} x_j = b_i r_i + \sum_{j=1}^{n} b_{ij} z_{ij} \quad i \in M \\ & z_{ij} \geq r_i - \overline{r}_i(1 - x_j) \quad i \in M,\, j \in N \ (1) \\ & z_{ij} \leq r_i \quad i \in M,\, j \in N \ (2) \\ & z_{ij} \leq \overline{r}_i x_j \quad i \in M,\, j \in N \ (3) \\ & x_j \in \{0, 1\},\, z_{ij} \geq 0 \quad i \in M,\, j \in N \end{cases}$$

où $\overline{r}_i$ est une borne supérieure de r_i.

Remarquons que les trois contraintes (1)-(3) ne sont pas toutes nécessaires pour imposer à la variable z_{ij} d'être égale au produit $r_i x_j$. On distinguera 2 cas : $b_{ij} > 0$ et $b_{ij} < 0$, ce qui donne le théorème suivant.

Théorème 2 : Soit (x^*, y^*, z^*) une solution optimale de $LIN2$. Alors x^* est une solution optimale de SRH.

$$(LIN2) \begin{cases} Max & \sum_{i=1}^{m} r_i \\ s.c. & a_i + \sum_{j=1}^{n} a_{ij} x_j = b_i r_i + \sum_{j=1}^{n} b_{ij} z_{ij} \quad i \in M \\ & z_{ij} \geq r_i - \overline{r}_i(1 - x_j) \,\big|\, si\, b_{ij} > 0 \quad i \in M,\, j \in N \\ & \left.\begin{array}{l} z_{ij} \leq r_i \\ z_{ij} \leq \overline{r}_i x_j \end{array}\right| si\, b_{ij} < 0 \quad i \in M,\, j \in N \\ & x_j \in \{0, 1\},\, z_{ij} \geq 0 \quad i \in M,\, j \in N \end{cases}$$

Preuve : On va montrer que $z_{ij} = r_i x_j$ à l'optimum de $LIN2$.

* Supposons $x \in \{0, 1\}^n$ fixé, considérons un i quelconque de M et notons $I_1 = \{j \in N : x_{ij} = 1\}$. On a :

1. Pour $b_{ij} > 0$.

- Si $x_j = 0$, en relachant la ijeme contrainte de type (1) on aura :

$$z_{ij} \geq 0 \Longrightarrow z_{ij} = 0 + \varepsilon_{ij},\, \varepsilon_{ij} \geq 0$$

- Si $x_j = 1$, alors d'après (1) on a :

$$z_{ij} \geq r_i \Longrightarrow z_{ij} = r_i + \varepsilon_{ij},\, \varepsilon_{ij} \geq 0$$

2. Pour $b_{ij} < 0$
- Si $x_j = 0$, alors d'après (3) on a :

$$z_{ij} \leq 0 \Longrightarrow z_{ij} = 0 - \varepsilon_{ij},\, \varepsilon_{ij} \geq 0$$

- Si $x_j = 1$, en relachant la ijeme contrainte de type (3) on aura :

$$z_{ij} \leq r_i \,(\text{d'après la contrainte (2)}) \Longrightarrow z_{ij} = r_i - \varepsilon_{ij}$$

En reportant dans l'égalité du modèle $LIN2$, on obtient :

$$a_i + \sum_{j \in I_1} a_{ij} = b_i r_i + \sum_{j \in I_1} b_{ij} r_i + \underbrace{\sum_{j:b_{ij}>0} b_{ij}\varepsilon_{ij} - \sum_{j:b_{ij}<0} b_{ij}\varepsilon_{ij}}_{c(\varepsilon)\geq 0}$$

$$\Rightarrow a_i + \sum_{j \in I_1} a_{ij} = (b_i + \sum_{j \in I_1} b_{ij}) r_i + c(\varepsilon)$$

$$\Rightarrow \frac{a_i + \sum_{j \in I_1} a_{ij}}{b_i + \sum_{j \in I_1} b_{ij}} = r_i + \frac{c(\varepsilon)}{b_i + \sum_{j \in I_1} b_{ij}}$$

alors r_i est borné supérieurement par la constante $R_i = \dfrac{a_i + \sum_{j \in I_1} a_{ij}}{b_i + \sum_{j \in I_1} b_{ij}}$. Donc, la valeur de $LIN2 \leq \sum_{i=1}^{m} R_i$ qui est égale à la valeur de SRH'' pour tout $x \in \{0,1\}^n$ (I).

* Maintenant, soit $\tilde{x}$ une solution optimale de SRH''.
On construit une solution $(\tilde{x}, \hat{r}, \hat{z})$ admissible pour $LIN2$ ayant la même valeur de fonction objectif en posant $\hat{r}_i = \left(a_i + \sum_{j=1}^{n} a_{ij}\tilde{x}_j\right) \Big/ \left(b_i + \sum_{j=1}^{n} b_{ij}\tilde{x}_j\right)$ et $\hat{z}_{ij} = \hat{r}_i \tilde{x}_{ij}\, \forall\, i \in M$ et $j \in N$. Donc, la valeur de la solution optimale de $LIN2$ est supèrieure ou égale à la valeur optimale de SRH''(II).

D'après (I) et (II), on déduit que l'optimum de $LIN2$ est égal à l'optimum de SRH''. Ceci est vrai, si et seulement si, pour tout $i \in M$ $r_i = R_i$, c'est-à-dire ssi $\varepsilon_{ij} = 0 \, \forall i, j$. Ainsi $z_{ij} = x_j r_i$ pour toute solution optimale de $LIN2$.

3.3.1.3 Amélioration de la linéarisation de Wu

Pour améliorer la linéarisation $LIN1$, nous considérons à la fois une borne supérieure $\overline{y}_i$ et une borne inférieure $\underline{y}_i$ 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 degageons une partie linéaire $(\underline{y}_i x_j)$ dans la fonction objectif et dans les containtes. Nous obtenons donc :

$$
(LIN3) \left\{ \begin{array}{lll}
Max & \displaystyle\sum_{i=1}^{m}(a_i y_i + \sum_{j=1}^{n} a_{ij} z_{ij} + \sum_{j=1}^{n} a_{ij} \underline{y}_i x_j) & \\
s.c. & b_i y_i + \displaystyle\sum_{j=1}^{n} b_{ij} z_{ij} + \sum_{j=1}^{n} b_{ij} \underline{y}_i x_j = 1 & i \in M \\
 & z_{ij} \geq y_i - \bar{y}_i(1 - x_j) - \underline{y}_i x_j & i \in M,\, j \in N \ (1) \\
 & z_{ij} \leq y_i - \underline{y}_i & i \in M,\, j \in N \ (2) \\
 & z_{ij} \leq (\bar{y}_i - \underline{y}_i) x_j & i \in M,\, j \in N \ (3) \\
 & x_j \in \{0,\, 1\} \,,\, z_{ij} \geq 0 & i \in M,\, j \in N
\end{array} \right.
$$

La formulation $LIN3$ 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 2 : La valeur optimale de $\overline{LIN3}$, la relaxation continue de $LIN3$, est inférieure ou égale à la valeur optimale $\overline{LIN1}$, la relaxation continue de $LIN1$.

Preuve : Soit $(\tilde{x},\, \tilde{y},\, \tilde{z})$ une solution optimale de $\overline{LIN3}$. On construit une solution $(\tilde{x},\, \tilde{y},\, \hat{z})$ admissible pour $\overline{LIN1}$ ayant la même valeur de fonction objectif en posant $\hat{z}_{ij} = \tilde{z}_{ij} + \underline{y}_i \tilde{x}_j \, \forall \, i \in M$ et $j \in N$.

3.3.1.4 Amélioration de la nouvelle linéarisation

L'idée de cette linéarisation est similaire à celle que nous avons utilisée pour $LIN3$. Nous exploitons la connaissance d'une borne inférieure $\underline{r}_i$ de r_i. z_{ij} remplacera le produit $(r_i - \underline{r}_i)x_j$

au lieu du produit $r_i x_j$. Le programme obtenu est le suivant :

$$(LIN4)\begin{cases} Max & \sum_{i=1}^{m} r_i \\ s.c. & a_i + \sum_{j=1}^{n} a_{ij}x_j = b_i r_i + \sum_{j=1}^{n} b_{ij} z_{ij} + \sum_{j=1}^{n} b_{ij}\underline{r}_i x_j \quad i \in M \\ & z_{ij} \geq r_i - \overline{r}_i(1-x_j) - \underline{r}_i x_j |\, si\, b_{ij} > 0 \quad i \in M,\, j \in N \ (1) \\ & \left.\begin{array}{l} z_{ij} \leq r_i - \underline{r}_i \\ z_{ij} \leq (\overline{r}_i - \underline{r}_i)x_j \end{array}\right| si\, b_{ij} < 0 \quad i \in M,\, j \in N \ (2)\, et\, (3) \\ & x_j \in \{0,1\},\, z_{ij} \geq 0 \quad i \in M,\, j \in N \end{cases}$$

$LIN4$ comprend le même nombre de variables et de contraintes que $LIN2$, mais nécessite le calcul de bornes inférieures de r_i.

Proposition 3 : La valeur optimale de $\overline{LIN4}$, la relaxation continue de $LIN4$, est inférieure ou égale à la valeur optimale $\overline{LIN2}$, la relaxation continue de $LIN2$.

Preuve : La preuve est similaire à la preuve de la proposition 2.

Le tableau 3.1 résume le nombre de variables et de contraintes de chaque linéarisation.

Linéarisation	Nombre de variables	Nombre de contraintes
$LIN1$	$n+m+mn$	$m+3mn$
$LIN2$	$n+m+mn$	$m+2mn$
$LIN3$	$n+m+mn$	$m+3mn$
$LIN4$	$n+m+mn$	$m+2mn$

TAB. 3.1 – Comparaison des modèles du point de vue nombre de variables et nombre de contraintes (n : nombre de ratios et m le nombre de ratios)

3.3.2 Élaboration des bornes supérieures et inférieures

Les différents modèles proposés nécessitent la connaissance de bornes inférieures et supérieures de certaines variables. La difficulté de calcul de ces bornes diffère d'une linéarisation à l'autre. Pour les linéarisations $LIN1$ et $LIN3$, on dispose de bornes évidentes pour y_i. On peut en effet prendre comme borne supérieure de y_i, $\overline{y}_i = 1/b_i + \sum_{j=1:b_{ij}<0}^{n} b_{ij}$ et comme borne inférieure, $\underline{y}_i = 1 \Big/ \left(b_i + \sum_{j=1:b_{ij}>0}^{n} b_{ij}\right)$.

Le calcul de $\overline{r}_i$ et $\underline{r}_i$, les bornes de r_i utilisées dans $LIN2$ et $LIN4$ nécessite davantage d'effort. Nous avons résolu pour tout $i \in M$, les deux problèmes hyperboliques en 0/1 sans contraintes : $Max/Min \left\{ \dfrac{a_i + \sum_{j=1}^{n} a_{ij} x_j}{b_i + \sum_{j=1}^{n} b_{ij} x_j} : x_j \in \{0, 1\}^n \right\}$ pour lesquels, comme nous l'avons déjà évoqué dans la section 3.2, il existe plusieurs méthodes de résolution. Nous avons choisi d'utiliser une approche par paramétrisation, plus précisément une méthode Newtonienne, l'algorithme 8 de Isbell et Marlow [Isbell et al. 1956]. Rappelons qu'il s'agit d'un algorithme itératif qui demande, à chaque itération, l'optimisation d'une fonction linéaire sans contraintes.

3.3.3 Expériences numériques et comparaisons

Tous les programmes associés aux différentes linéarisations présentées ont éte résolus en utilisant CPLEX 6.0 [Cplex, 1998] sur un ordinateur de type Pentium II à 300 MHz. Les expérimentations ont été réalisées sur des problèmes engendrés aléatoirement. Nous avons considéré deux types d'instances. Pour le premier type, les coefficients b_i, a_{ij} et b_{ij} de la fonction objectif sont des entiers uniformément distribués entre 1 et 1000 et les coefficients a_i sont des entiers uniformément distribués entre 1 et 100. Pour le second type d'instances, qui se sont avérées plus difficiles, les coefficients b_i, a_{ij} et b_{ij} sont choisis uniformément entre 10 et 20 et les coefficients a_i, entre 1 et 10.

	coefficients a_i	coefficients b_i, a_{ij}, b_{ij}
type1	[1, 100]	[1, 1000]
type2	[1, 10]	[10, 20]

Les tableaux 3.2 et 3.3 comparent les valeurs optimales des relaxations continues des programmes $LIN1$, $LIN2$, $LIN3$ et $LIN4$ pour les deux types d'instances. Pour chaque relaxation nous donnons la valeur de la solution (val) et le temps de résolution (Tps).

D'après les tableaux 3.2 et 3.3, il est clair que la relaxation du programme $LIN3$ est la plus précise et que la relaxation de $LIN1$ est la moins précise. Néanmoins, $LIN3$ est-elle la linéarisation la plus efficace et $LIN1$ la moins efficace pour résoudre le problème SRH ?

Nous présentons dans les tableaux 3.4 et 3.5 les résultats de la résolution exacte pour les quatre linéarisations. Le premier tableau concerne le premier type d'instances et le second, le deuxième type d'instances. Chaque ligne des deux tableaux représente la moyenne des

Problème		$\overline{LIN1}$		$\overline{LIN2}$		$\overline{LIN3}$		$\overline{LIN4}$	
m	**n**	**val**	**Tps (s)**	**val**	**Tps (s)**	**val**	**Tps (s)**	**val**	**Tps (s)**
10	30	39.82	0.43	37.09	0.19	**32.10**	**0.51**	36.85	0.21
10	30	37.68	0.35	33.59	0.20	**30.24**	**0.37**	33.47	0.18
10	30	41.49	0.24	36.52	0.20	**32.06**	**0.3**	36.36	0.22
20	50	137.96	2.41	125.89	1.47	**104.53**	**5.11**	124.89	1.61
20	50	89.52	2.07	81.92	1.96	**67.29**	**3.29**	80.55	2.46
20	50	105.32	2.11	97.11	1.06	**81.33**	**2.36**	96.31	1.26
30	50	134.99	6.69	124.68	4.74	**99.96**	**10.77**	124.11	4.72
30	50	175.37	4.34	161.70	4.28	**133.18**	**11.25**	162.26	4.88
30	50	139.61	3.47	128.17	4.74	**102.86**	**7.67**	127.24	4.44

TAB. 3.2 – Résultats numériques associés aux relaxations continues des 4 linéarisations en variables mixtes du problème SRH (premier type d'instances)

Problème		$\overline{LIN1}$		$\overline{LIN2}$		$\overline{LIN3}$		$\overline{LIN4}$	
m	**n**	**val**	**Tps (s)**	**val**	**Tps (s)**	**val**	**Tps (s)**	**val**	**Tps (s)**
10	30	13.25	0.30	12.89	0.20	**12.36**	**0.28**	12.86	0.25
10	30	12.78	0.28	12.33	0.19	**11.98**	**0.30**	12.31	0.22
10	30	12.81	0.20	12.40	0.24	**11.96**	**0.27**	12.36	0.21
20	50	27.53	3.43	26.89	2.19	**25.6**	**3.91**	26.79	1.95
20	50	27.95	3.72	27.3	1.76	**26.1**	**3.53**	27.24	1.72
20	50	27.34	2.49	26.77	1.96	**25.69**	**3.36**	26.66	2.73
30	50	41.88	12.52	40.92	3.91	**39.02**	**6.86**	40.81	4.50
30	50	40.62	7.34	39.66	4.01	**39.53**	**9.40**	39.53	4.59
30	50	41.05	4.45	40.07	3.80	**39.95**	**8.24**	39.95	4.42

TAB. 3.3 – Résultats numériques associés aux relaxations continues des différentes linéarisations en variables mixtes du problème SRH (deuxième type d'instances)

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 10000 secondes par instance, la moyenne porte sur les seules instances résolues exactement. Pour chaque type de problème, nous indiquons le nombre de nœuds développés dans le branch-and-bound (Nb_nd), la durée totale de résolution en secondes (Tps) ainsi que le nombre d'instances résolues (#) de façon exacte.

Prob		$LIN1$			$LIN2$			$LIN3$			$LIN4$		
m	**n**	**Nb_nd**	**Tps**	#	**Nb_nd**	**Tps**	#	**Nb_nd**	**Tps**	#	**Nb_nd**	**Tps**	#
10	10	297	0.9	5	408	1.0	5	**133**	**0.7**	**5**	382	1.0	5
10	20	9792	42.7	5	26286	10.5	5	**1861**	**17.4**	**5**	22109	88.9	5
10	30	87647	582.7	5	18721	251.9	2	**17168**	**279.8**	**5**	17940	267.0	2
20	10	178	1.4	5	246	1.5	5	**74**	**1.2**	**5**	208	1.3	5
20	20	78342	636.9	5	234175	2173.7	5	**25498**	**551.9**	**5**	208369	2003.1	5
30	10	653	5.4	5	745	7.5	5	**258**	**4.7**	**5**	739	7.5	5
30	20	90305	1933.4	5	261088	5703.7	5	**26429**	**1651.2**	**5**	250384	5076.1	5

TAB. 3.4 – Comparaison numérique de la résolution exacte du problème par les 4 linéarisations présentées (premier type d'instances)

Prob		$LIN1$			$LIN2$			$LIN3$			$LIN4$		
m	**n**	**Nb_nd**	**Tps**	#	**Nb_nd**	**Tps**	#	**Nb_nd**	**Tps**	#	**Nb_nd**	**Tps**	#
10	10	336	1.0	5	303	0.7	5	**102**	**0.5**	**5**	292	0.7	5
10	20	20978	101.1	5	165259	687.4	5	**4582**	**39.1**	**5**	106695	400.1	5
10	30	313456	4322.6	1	-	-	0	**379377**	**5223.4**	**3**	-	-	0
20	10	514	2.8	5	547	2.5	5	**127**	**1.6**	**5**	430	2.3	0
20	20	136347	1343.4	5	-	-	0	**29239**	**774.7**	**5**	276657	2463.1	2
30	10	575	5.1	5	616	4.4	5	**142**	**3.6**	**5**	475	4.0	5
30	20	240881	3755.1	4	-	-	0	**61988**	**3232.5**	**5**	-	-	0

TAB. 3.5 – Comparaison numérique de la résolution exacte du problème par les linéarisations présentées (deuxième type d'instances)

Les tableaux 3.4 et 3.5 montrent que la linéarisation $LIN3$ est la plus efficace, ce qui confirme les résultats obtenus par la relaxation continue. En revanche, ce sont les linéarisations $LIN2$ et $LIN4$ qui sont les moins efficaces et non pas $LIN1$, bien que celle-ci donne des bornes moins précises. Pour $LIN2$ et $LIN4$, nous avons constaté que la convergence vers la solution optimale était très lente.

On constate également que le deuxième type d'instances est beaucoup plus difficile à résoudre et que nous n'avons pu résoudre que des problèmes de taille moyenne.

Il est clair aussi que le temps de résolution croît considérablement avec la taille des problèmes, ce qui ne permet pas la résolution de grandes 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 (voir table 3.6). L'écart entre la valeur de la relaxation continue du problème et la valeur de la solution optimale ($\frac{\overline{LIN_i} - OPT}{OPT}$, $i = 1, ..., 4$) est très important (au moins 68% pour $LIN1$ et 38% pour $LIN3$).

mxn	$\overline{LIN1}$	$\overline{LIN3}$	OPT	$\frac{\overline{LIN1} - OPT}{OPT}$ (%)	$\frac{\overline{LIN3} - OPT}{OPT}$ (%)
10x10	23.54	19.38	14.01	68	38
10x30	36.68	30.75	16.61	120.8	85
20x10	51.67	41.84	24.78	108.5	68.8
30x10	77.82	60.89	36.19	115	68.25

Tab. 3.6 – Comparaison entre la valeur de la relaxation continue et la valeur de la solution exacte pour $LIN1$ et $LIN3$ (moyenne de 5 instances)

Le tableau 3.6 confirme également que l'écart relatif est moins important pour la linéarisation $LIN3$. Par exemple on passe de 68% à 38% et de 115% à 68.25%. En effet, les coefficients de $LIN3$ sont plus fins que ceux de $LIN1$.

Pour améliorer la résolution du problème SRH, nous proposons ci-après une approche heuristique permettant de résoudre rapidement, de manière approchée, de grandes instances. Cette solution heuristique sera également exploitée pour raffiner les bornes des variables y_i et r_i afin d'améliorer la résolution exacte.

3.4 Une méthode heuristique

3.4.1 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.

Les expérimentations ont montré que la solution admissible obtenue par cette heuristique

Algorithm 12 Approche Heuristique

1. S_0 = (0, 0, ..., 0). $FO = \sum_{i=1}^{m} \frac{a_i}{b_i}$ (FO : valeur de la fonction objectif)
2. Pour chaque $x_i = 0$ (respectivement $x_i = 1$) calculer la valeur de la fonction objectif en fixant x_i à 1 (respectivement x_i à 0) dans la dernière solution trouvée.
3. Sélectionner la meilleure solution améliorant la dernière solution trouvée. Mise à jour de la solution et de FO.
4. Si aucune solution n'améliore la dernière solution trouvée, stop sinon aller à 2.

était de très bonne qualité. Dans la plupart des cas, elle est égale à la solution optimale. Elle est également très utile pour la résolution exacte. En effet, la connaissance d'une bonne solution admissible permet d'améliorer la résolution directe des différents programmes proposés. En fournissant la borne inférieure correspondant à cette solution admissible au logiciel de programmation linéaire en variables mixtes, nous obtenons une réduction significative de l'arbre d'exploration. Des résultats numériques montrant son efficacité sont présentés au paragraphe suivant.

3.4.2 Résultats expérimentaux

Afin de déterminer une solution approchée, nous avons implémenté en langage C sur un Pentium II à 300 MHz l'heuristique présentée ci-dessus. Les tests ont été réalisés avec le deuxième type d'instances qui sont décrites dans le paragraphe 3.3.3. Ces tests concernent uniquement l'heuristique proposée et son influence sur la résolution du problème $LIN3$, celle qui a déjà donné les meilleurs résultats (voir paragraphe 3.3.3). Le tableau 3.7 présente les résultats correspondants. Chaque ligne du tableau correspond à 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 10000 secondes par instance, la moyenne porte sur les seules instances résolues exactement. Nous indiquons, pour l'heuristique, le nombre d'instances pour lesquelles on obtient la solution optimale si cette dernière est connue (#) et le temps de calcul (Tps). Pour $LIN3$ avec et sans la valeur de la solution de l'heuristique (solution initiale), nous précisons le nombre de nœuds (Nb_nd), le temps CPU (Tps) et le nombre d'instances que nous avons pu résoudre (#).

Le tableau 3.7 montre que notre heuristique est très efficace. Elle donne dans la plupart des cas la solution optimale en un temps excessivement court, moins d'une seconde quelle que soit la taille du problème. On constate également qu'en donnant la valeur de la solution

		Cplex sans sol_initiale					**Cplex avec sol_initiale**		
Problème		$LIN3$			**Heuristique**		$LIN3$		
m	**n**	**Nb_nd**	**Tps (s)**	**#**	**Tps (s)**	**#s**	**Nb_nd**	**Tps (s)**	**#**
5	10	41	0.1	5	0.00039	5	32	0.08	5
5	20	1088	3.9	5	0.00058	5	921	3.1	5
5	30	67177	324.5	5	0.00097	5	56631	236.2	5
5	40	184667	1154.5	2	0.001952	3	680502	3200.6	3
10	10	141.2	0.6	5	0.00097	5	131	0.58	5
10	20	20575	147.2	5	0.00194	5	18302	127.0	5
10	30	340451	4379.2	3	0.00167	4	400512	4374.1	4
20	10	95	1.4	5	0.00178	5	91	1.4	5
20	20	37281	880.6	5	0.00488	5	33852	762.2	5

TAB. 3.7 – Comparaison numérique entre $LIN3$, l'heuristique proposée et $LIN3$ avec la valeur de la solution heuristique

obtenue par l'heuristique au solveur, le nombre de nœuds de l'arbre d'exploration est réduit. Ainsi, le temps de résolution est moins important (voir les colonnes $LIN3$ avec et sans solution initiale).

Dans le but d'améliorer la résolution exacte des linéarisations présentées, nous proposons dans le paragraphe suivant une méthode de raffinement des bornes des variables y_i et r_i permettant d'accélérer la procédure de recherche de l'optimum. Cette méthode exploite la borne inférieure donnée par la méthode heuristique.

3.5 Raffinement des coefficients des modèles de PLVM et résultats expérimentaux

Nous montrons dans cette partie comment obtenir des bornes inférieures et supérieures plus précises pour les variables y_i et r_i, $i \in M$. Le paragraphe 3.5.1 décrit la démarche adoptée. Les résultats numériques prouvant l'intérêt de la méthode sont présentés au paragraphe 3.5.2.

3.5.1 Description de la méthode

Nous avons proposé au paragraphe 3.3.2 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 $B_k = \sum_{i=1}^{m} \left(Max \left\{ \frac{a_i + \sum_{j=1}^{n} a_{ij} x_j}{b_i + \sum_{j=1}^{n} b_{ij} x_j} : x \in \{0,1\}^n, \sum_{j=1}^{n} x_j = k \right\} \right)$. 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 $X = \{x : x \in \{0,1\}^n, \sum_{i=1}^{n} x_i \leq K_1, \sum_{i=1}^{n} x_i \geq K_2\}$. $\underline{y}_i$ et $\overline{y}_i$ sont alors données par la resolution des 2 programmes hyperboliques sous contraintes $Max/Min \left\{ 1 \Big/ \left(b_i + \sum_{j=1}^{n} b_{ij} x_j \right) : x \in X \right\}$, et $\underline{r}_i$ et $\overline{r}_i$ sont données par la resolution des 2 programmes $Max/Min \left\{ \left(a_i + \sum_{j=1}^{n} a_{ij} x_j \right) \Big/ \left(b_i + \sum_{j=1}^{n} b_{ij} x_j \right) : x \in X \right\}$. 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.

3.5.2 Résultats Numériques

Nous présentons dans le tableau 3.8 des résultats numériques permettant de comparer la résolution par $LIN3$ couplée avec l'heuristique avec ou sans l'étape de raffinement. Les expérimentations ont été effectuées avec les mêmes instances que celles qui ont été décrites au paragraphe 3.4.2 et également avec des instances de plus grande taille engendrées de la même façon.

Chaque ligne du tableau présente la moyenne des résultats concernant 5 instances générées aléatoirement. Pour chaque type de résolution, nous indiquons le nombre de nœuds développés dans le branch-and-bound (Nb_nd), la durée de la résolution (Tps) ainsi que le nombre d'instances résolues parmi les 5 considérées (#). Le temps de résolution d'une instance est limité à 10000 secondes.

Il est clair d'après le tableau 3.8 que l'étape de raffinement est très efficace. En effet, en améliorant les bornes des variables y_i dans $LIN3$, nous obtenons une réduction de l'arbre

Problème		$LIN3$ **sans raffinement**			$LIN3$ **avec raffinement**		
m	**n**	**Nb_nd**	**Tps (s)**	#	**Nb_nd**	**Tps (s)**	#
5	10	32	0.1	5	**23**	**0.07**	**5**
5	20	921	3.1	5	**490**	**1.9**	**5**
5	30	56631	236.2	5	**31823**	**137.2**	**5**
10	10	131	0.58	5	**111**	**0.5**	**5**
10	20	18302	127.0	5	**14491**	**98.6**	**5**
10	30	400511	4374.1	4	**210677**	**2463.9**	**4**
20	10	91	1.4	5	**57**	**1.1**	**5**
20	20	33851	762.2	5	**25626**	**697.5**	**5**
4	50	216832	1171.5	5	**73695**	**401.0**	**5**
2	100	288901	1252.9	4	**181902**	**404.8**	**5**

TAB. 3.8 – Comparaison numérique entre ($LIN3$) avec et sans raffinement

d'exploration et une accélération de la résolution. Par exemple, pour les instances (10x30), $LIN3$ sans raffinement permet de résoudre 4 instances en un temps moyen de 4400 secondes, alors que $LIN3$ avec raffinement permet de résoudre 4 instances en moins de 2500 secondes. On note également que plus le nombre de variables est grand, plus l'étape de raffinement est efficace (voir les instances 4x50 et 2x100).

3.6 Un algorithme Branch-and-bound avec réajustement progressif des coefficients des modèles

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 modèles proposés pour l'efficacité de la résolution par un logiciel standard. C'est pourquoi nous proposons maintenant un algorithme 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.

3.6.1 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 branch-and-bound standard, l'unique différence entre le sous-problème à résoudre dans un nœud père et celui à résoudre dans un nœud fils est le remplacement d'une seule contrainte ($0 \leq x_i \leq 1$) par une contrainte de fixation ($x_i = 1$ ou $x_i = 0$). Dans notre procédure spécifique, certains coefficients du modèle vont évoluer 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.

En chaque nœud de l'arbre de recherche est évaluée une borne supérieure (UB) du problème, obtenue par relaxation continue. Si UB <LB (la valeur de la meilleure solution trouvée), la branche de l'arbre de recherche est coupée. Sinon, une nouvelle variable de branchement est choisie. Cette variable est fixée à 0 puis à 1 ou bien à 1 puis à 0 selon sa valeur dans la solution admissible (solution heuristique). Au niveau de chaque nouveau branchement, on réajuste les coefficients ($\underline{y}_i$, $\overline{y}_i$) ou ($\underline{r}_i$, $\overline{r}_i$) en fonction des variables déjà fixées dans la branche. Pour cela, on fait appel à la procédure d'élaboration des bornes inférieures et supérieures ($\underline{y}_i$, $\overline{y}_i$, $\underline{r}_i$, $\overline{r}_i$), présentée au paragraphe 3.3.2, en ajoutant des contraintes sur les variables dèjà 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 pour l'étape d'évaluation en chaque nœud est plus important pour notre procédure. Contrairement à un algorithme 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 réajustement 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 n'est pas intéressant de réajuster les coefficients si le nombre de variables fixées est encore petit, l'amélioration des coefficients risquant d'être faible et la coupe étant peu probable.

3.6.2 Résultats numériques et comparaisons

Nous indiquons dans le tableau 3.9 les résultats concernant 4 approches pour la résolution du problème SRH : $LIN3$, $LIN4$ avec un branch-and-bound standard couplée avec l'heuristique et l'étape de raffinement (CPLEX+Sol_ Heuristique+Raf), puis $LIN3$ et $LIN4$ 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 décrites dans le paragraphe 3.4.2. Pour chaque type de résolution, nous indiquons le nombre total de nœuds (Nb_nd) et le temps CPU (Tps).

		CPLEX+Sol_ Heuristique+Raf				**Branch&Bound Spécifique**			
Prob		$LIN3$		$LIN4$		$LIN3$		$LIN4$	
m	**n**	**Nb_nd**	**Tps**	**Nb_nd**	**Tps**	**Nb_nd**	**Tps**	**Nb_nd**	**Tps**
10	20	5736	46.7	35142	170.5	2089	64.0	**2311**	**29.5**
10	20	5797	49.1	29806	153.2	1843	57.2	**2709**	**24.2**
10	20	2106	18.0	27296	166.6	1421	38.6	**1179**	**14.1**
20	20	24894	705.9	81460	1088.1	4045	497.6	**5337**	**185.4**
20	20	24715	616.7	76610	1021.4	3199	399.5	**4161**	**142.1**
20	20	40956	1034.6	99610	1440.2	4627	522.6	**6917**	**228.1**
30	20	35315	1869.5	95160	3628.4	4143	961.8	**5267**	**356.6**
30	20	17559	1019.3	79321	2063.4	3609	836.9	**4613**	**310.3**
30	20	15058	972.6	26838	876.6	3001	748	**3183**	**270.6**

TAB. 3.9 – Comparaison numérique entre le branch-and-bound standard (CPLEX) et notre procédure spécifique

Le tableau 3.9 montre que la résolution de $LIN3$ et $LIN4$ avec la procédure spécifique proposée est très efficace par rapport à la procédure standard. Elle permet de résoudre plus rapidement toutes les instances testées. Le nombre total de nœuds est de 3 à 4 fois plus petit que dans le cas du branch-and-bound standard (CPLEX).

On note également que, contrairement à la résolution standard où le programme $LIN3$ donne toujours les meilleurs résultats, c'est $LIN4$ qui fournit les meilleurs résultats dans le cas du branch-and-bound spécifique. En utilisant le branch-and-bound standard, $LIN4$ 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 $LIN3$. Nous pouvons donc conclure que $LIN4$ est beaucoup plus sensible aux valeurs des bornes que $LIN3$.

3.7 Conclusion

Nous avons consideré dans ce chapitre 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 pour lequel il existe très peu de travaux. Nous avons proposé différentes techniques de linéarisations qui se distinguent par les expressions linéarisées et les contraintes ajoutées. Ces techniques sont très faciles à mettre en œuvre lorsque l'on dispose d'un outil de programmation linéaire en variables mixtes. De plus, elles pourraient être appliquées au cas de problèmes sous contraintes ainsi qu'à de nombreux autres problèmes classiques de programmation quadratique en variables mixtes. Nous avons également proposé trois stratégies de résolution. La première consiste à résoudre directement le problème en utilisant un outil de programmation linéaire mixte. La deuxième stratégie est une résolution heuristique qui est aussi exploitée pour améliorer la première stratégie. Enfin, la troisième consiste à résoudre le problème par un algorithme branch-and-bound spécifique, fondé sur une réécriture plus précise des programmes 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). Toutes ces techniques de linéarisations et stratégies ont été comparées expérimentalement. Ces comparaisons ont montré que la résolution du problème par le branch-and-bound spécifique appliqué à l'une des trois 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.

Bibliographie

[Arora et al., 1977] S.R. Arora, K. Swarup, M.C. Puri, 'The set overing problem with linear fractional functional', Indian Journal of Pure and Applied Mathematics, Vol. 8, p. 578-588 (1977).

[Billionnet, 2002] A.Billionnet, 'Approximation algorithms for fractional knapsack problems', Operations Research Letters vol. 30, p. (2002).

[Billionnet et al., 2003] A. Billionnet, K. Djebali, 'Maximisation sans contraintes de la somme de plusieurs ratios hyperboliques en 0-1', In Proceeding ROADEF'03, Avignon (2003).

[Billionnet et al., 2003c] A. Billionnet, K. Djebali, 'Résolution d'un problème combinatoire fractionnaire par la programmation linéaire mixte', Majecstic'03 Proceeding, p. (2003).

[Charnes et al., 1962] A. Charnes, W.W. Cooper, 'Programming with Linear Fractional Functionals', Naval Research Logistic Quarterly, Vol. 9, p. 181-186 (1962).

[Chen et al., 2000] D.Z. Chen, O. Daescu, Y. Dai, N. Katoh, X. Wu, J. Xu, 'Efficient Algorithms and Implementations for Optimizing the Sum of Linear Functions with Applications', Proceedings of the 11^{th} annual SIAM-ACM Symposuim on Discrete Algorithms (SODA), San Francisco, p. 707-716 (2000).

[Cplex, 1998] Ilog, cplex.

[Dinkelbach, 1967] W. Dinkelbach, 'On nonlinear fractional programming', Management Science, Vol. 13, p. 492-498 (1967).

[Falk et al., 1992] J.E. Falk, S.W. Paloscay, 'Optimising the sum of Linear Fractional Functions', Advances in Global Optimisation, Kluwer Academic Publishers, p. 221-258 (1992).

[Freund et al., 2001] R.W. Freund, F. Jarre, 'Solving the sum-of-ratios problem by an interior-point method', Journal of Global Optimization, Vol. 19, p. 83-102 (2001).

[Gilmore et al., 1963] P.C. Gilmore, R.E. Gomory, 'A linear programming approach to the cutting stock problem', Operations Research Vol. 11, p. 52-53 (1963).

[Hammer et al., 1968] P.L. Hammer, S. Rudeanu, 'Boolean methods in operations research and related areas', Springer, Berlin (1968).

[Hansen et al., 1991] P. Hansen, M. V. Poggi De Aragao, C.C. Ribeiro, 'Hyperbolic 0-1 programming and information retrieval', Mathematical Programming, Vol. 52, p. 255-263 (1991).

[Hansen et al., 1987] P. Hansen, M.Minoux, M. Labbé, 'Extension de la programmation linéaire généralisée au cas des programmes mixtes', C.R. Acad. Sci. Paris, Série I Mathématiques, Vol. 305, p. 569-572 (1987).

[Ibaraki, 1983] T. Ibaraki, 'Parametric approaches to fractional programs', Mathematical Programming, Vol. 26, p. 345-362 (1983).

[Isbell et al. 1956] J. R. Isbell, W. H. Marlow, 'Attribution games', Naval Research Logistics Quarterly, Vol. 3, p. 71-94 (1956).

[Li, 1994] H. Li, 'A Global Approach for General Fractional Programming', European Journal of Operational Research, Vol. 73, p. 590-59 (1994).

[Meggido, 1979] N. Meggido, 'Combinatorial Optimization with Rational Objectives Functions', Mathematics of Operations Research, Vol. 4, p. 414-424 (1979).

[Nagih et al., 1995] A. Nagih, G. Plateau, 'An exact method for the 0-1 fractional knapsack problem', INFORMS, New Orleans, USA (1995).

[Nagih et al., 1999] A. Nagih, G. Plateau, 'Problèmes fractionnaires : Tour d'horizon sur les applications et méthodes de résolution', RAIRO Operations Research, Vol. 33, p. 383-419 (1999).

[Nagih et al., 2000] A. Nagih, G. Plateau, 'A partition algorithm for 0-1 unconstrained hyperbolic programs', Investigacion Operativa, Vol. 9, p. 167-178 (2000).

[Radzik, 1998] T. Radzik, 'Fractional Combinatorial Optimization', Handbook of Combinatorial Optimization, Paradaloas, Kluwer Academic Publishers, p. 429-478 (1998).

[Robbilard, 1971] P. Robbilard, '0-1 Hyperbolic Programming', Naval Research Logistics Quarterly Vol. 18, p. 47-58 (1971).

[Saipe, 1975] A. L. Saipe, 'Solving a 0-1 Hyperbolic Program by Branch and Bound', Naval Research Logistics Quarterly, Vol. 22, p. 397-416 (1975).

[Schaible, 1976] S. Schaible, 'Duality in Fractional Programming : a Unified Approach', Operational Research, Vol. 24, p. 452-461 (1976).

[Schaible, 1981] S. Schaible, 'Fractional Programming : Applications and Algorthms', European Journal of Operational Research, Vol. 17, p. 111-120 (1981).

[Schaible et al., 1983] S. Schaible, T. Ibaraki, 'Fractional Programming', European Journal of Operational Research, Vol. 12, p. 325-338 (1983).

[Schaible, 1995] S. Schaible, 'Fractional Programming' Handbook of Combinatorial Optimization, Kluwer Academic Publishers, p. 495-608 (1995).

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

[Wu, 1997] T. Wu, 'A note on a global Approach for General 0-1 Fractional Programming', European Journal of Operational Research, Vol. 101, p. 229-223 (1997).

Chapitre 4

Résolution du problème de stable de cardinal maximal dans un graphe

4.1 Introduction

Soit un graphe non orienté connexe $G = (V, E)$ où V est l'ensemble des sommets et E l'ensemble des arêtes. Un sous ensemble de sommets $I \subset V$ est dit ensemble stable s'il ne comprend que des sommets non adjacents deux à deux. Un ensemble stable est maximum (MS) (maximum stable set en anglais) si le cardinal de tout ensemble stable est inférieur ou égal au cardinal de cet ensemble. Le cardinal de MS est appelé nombre de stabilité de G et est noté $\alpha(G)$. De nombreux problèmes pratiques peuvent être modélisés comme des problèmes de stable maximum. Citons, par exemple, les problèmes de localisation, les problèmes de diagnostic (ou de détection de pannes), les problèmes de classification, les problèmes de reconnaissance de formes et les problèmes de sélection des données.

Le problème du stable maximum est NP-difficile [Garey et al., 1979] ; il s'agit d'un des problèmes les plus durs de l'optimisation combinatoire. Il est équivalent à d'autres problèmes de la théorie des graphes : par exemple, le problème de la clique de taille maximale (maximum clique : MC) et le problème du recouverment des arêtes par un ensemble de sommets de cardinalité minimale (minimum vertex cover). En notant par $\bar{G}$ le graphe complémentaire de G, pour tout sous ensemble $S \subset V$, les trois propositions suivantes sont équivalentes :

1. S un ensemble stable de G,
2. S est une clique de $\bar{G}$,
3. $V \setminus S$ est un recouvrement de G

Nous allons étudier la résolution de ce problème par la programmation linéaire en nombres binaires. La première partie de ce chapitre est consacrée à l'état de l'art concernant la résolution du MS ou des problèmes équivalents. Dans la deuxième partie, nous présentons différentes formulations du MS par des programmes mathématiques en 0-1 qui diffèrent par le nombre et la nature des contraintes. Puis, dans la partie 3, sont présentés les résultats numériques obtenus lors de la résolution de MS par les formulations proposées et par un outil de programmation linéaire mixte. Une étude comparative de ces résultats avec ceux de la littérature est également présentée.

4.2 État de l'art

Étant donnée l'importance théorique et pratique du problème Max Stable/Max Clique, de très nombreuses études ont été publiées sur ce sujet. Nous présentons ci-après une synthèse des principales méthodes de résolution en distinguant les méthodes exactes des méthodes approchées. Cette présentation est loin d'être exhaustive car les publications sont extrêmment abondantes depuis une trentaine d'années.

4.2.1 Méthodes exactes

Il existe un nombre important de publications dans la littérature concernant la résolution exacte du Max Stable /Max Clique. La plupart de ces travaux on été développés durant les années 80. En 1982, Loukakis et Tsourous [Loukakis et al., 1982] ont proposé un algorithme de recherche arborescente pour résoudre le problème MS. Puis en 1984, Ebenegger et al. [Ebenegger et al., 1984] ont proposé un autre algorithme, basé sur la relation existante entre la maximisation d'une fonction pseudo-booléenne et le nombre de stabilité dans un graphe. Cette méthode permet de traiter des graphes de 100 sommets.

Une contribution très importante pour la résolution du problème MC a été proposée par Balas et Yu [Balas et al., 1986]. C'est un algorithme d'énumération implicite original qui applique à chaque nœud de l'arbre de recherche une procédure en deux étapes. La première étape consiste à trouver un sous-graphe triangulé maximal D de G, puis, une clique maximale de D fournissant ainsi une borne inférieure pour le problème MC. À l'étape suivante, une approche heuristique de coloration est appliquée afin d'étendre le sous-graphe D à un autre sous-graphe plus grand ayant la même taille de clique maximale. L'objectif est de réduire le nombre de sous-problèmes générés en chaque nœud de l'arbre d'exploration. Cette approche a permis la résolution de problèmes de grande taille allant jusqu'à 400 sommets et de densité allant jusqu'à

75%.

En 1986, Kikustus [Kikustus, 1986] a proposé un algorithme branch-and-bound basé sur une nouvelle relation récursive, pour le nombre de stabilité d'un graphe G, définie par : $\alpha(G) = max\left\{1+\alpha\left(G\setminus[\{v\}\cup N(v)]\right), \alpha_v^{'}\right\}$ avec $\alpha_v^{'} = max\left\{|I| : I \subseteq G\setminus\{v\}\right.$un ensemble indépendant avec$\left.|I\cap N(v)| \geq 2\right\}$. Cette relation est différente de la relation $\alpha(G) = max\left\{1+\alpha\left(G\setminus[\{v\}\cup N(v)]\right), \alpha\left(G\setminus\{v\}\right)\right\}$, traditionnellement, utilisée dans un algorithme branch-and-bound pour le problème MS. Elle est plus efficace mais nécessite plus d'effort pour l'implémentation. Quelques résultats numériques sont fournis dans [Kikustus, 1986] mais sans aucune comparaison avec d'autres algorithmes. En 1986 également, Robson [Robson, 1986] a proposé une version modifiée de l'algorithme récursif de Tarjan et Trajanowski [Tarjan et al., 1977]. Il a montré, à travers une analyse détaillée, que la complexité de son algorithme est de l'ordre de $O\left(2^{0.276n}\right)$, plus petite que la complexité $O\left(2^{n/3}\right)$ de l'algorithme proposé dans [Tarjan et al., 1977].

À la fin des années 1980, de nouveaux algorithmes ont été proposés dans [Gendreau et al. 1988] et [Tomita et al., 1988]. L'algorithme de Tomita et al. [Tomita et al., 1988] utilise un algorithme gourmand de coloration pour la recherche d'une borne supérieure pour le problème MC. Gendreau et al. [Gendreau et al. 1988] proposent un algorithme d'énumération implicite. Le type de branchement utilisé est basé sur le nombre de triangles auxquels appartient un sommet.

En 1990, plusieurs algorithmes ont été proposés [Pardalos et al., 1990], [Friden et al., 1989], [Carraghan et al., 1990] et [Babel et al., 1990]. Pardalos et Phillips [Pardalos et al., 1990] ont formulé le problème MC comme un problème d'optimisation globale quadratique indéfini avec des contraintes linéaires. Friden et al. [Friden et al., 1990] proposent un algorithme branch-and-bound pour le problème MS en utilisant les techniques de recherche tabou pour la recherche des bornes inférieure et supérieure. Carraghan et Pardalos [Carraghan et al., 1990] ont proposé un algorithme d'énumération implicite très efficace pour les graphes peu denses. Le point fort de cet algorithme réside dans le type de branchement utilisé. En utilisant cet algorithme, des problèmes de 500 sommets ont pu être résolus. Dans [Babel et al., 1990], Babel et Tinhofer proposent un algorithme branch-and-bound pour le problème MC. Pour la recherche de bornes supérieures et inférieures, ils ont utilisé une heuristique gloutonne de coloration de graphe bien connue, introduite par Brelaz [Brelaz, 1979], dite méthode de Saturation. Cette dernière est utilisée en chaque nœud de l'arbre de recherche. Cette étude mentionne des résultats numériques avec des graphes de 100 à 400 sommets pour différentes valeurs de densités (0.1 à 0.9). Dans [Babel, 1991], une amélioration de cet algorithme est proposée.

En 1991 , Xue [Xue, 1991] a proposé un algorithme branch-and-bound, pour le problème MC, fondé sur une nouvelle heuristique de coloration pour la recherche des bornes. Cette heuristique exploite le fait que la solution fractionnaire pour la coloration fournit une borne plus fine que la solution de la coloration entière pour le problème MC. Une réduction significative de la taille de l'arbre de recherche ainsi qu'une amélioration en temps de calcul sont constatées.

Dans [Della et al., 1994], Della Croce et Tadei ont formulé le problème MC comme un problème de sac à dos quadratique multivariables. Pour sa résolution, ils combinent un algorithme gourmand et une procédure branch-and-bound. Dans [Bourjolly et al., 1997], Bourjolly et al. proposent une méthode de génération par colonnes imbriquée dans un algorithme branch and bound pour le problème MC. Dans [Bourjolly et al. 1996], une nouvelle approche pour la minimisation d'une fonction quadratique en 0-1, relative à la satisfaction d'une séquence d'expressions booléennes a été utlisée pour la détermination de bornes inférieures. Cette dernière a été incorporée dans un algorithme branch-and-bound pour le problème MS et a été testée sur divers graphes du DIMACS benchmarck. Dans [Wood, 1997], un algorithme branch-and-bound a été proposé pour le problème MC. Différentes heuristiques de coloration pour la détermination des bornes infèrieure et supérieure ont été utlisées. Des expérimentations numériques, réalisées sur des graphes aléatoires avec différentes valeurs de densité et sur des graphes du DIMACS Benchamark, ont été présentées. En 1998, Sewell [Sewell, 1998] a proposé également un algorithme branch-and-bound pour le problème MS. Cet algorithme est très efficace pour les graphes peu denses.

4.2.2 Méthodes approchées

Dans le but de trouver une solution de bonne qualité en un temps de calcul raisonnable, de nombreuses méthodes approchées ont été proposées pour la résolution du MS et MC. Parmi ces approches, on retrouve des méthodes gloutonnes, des méthodes de recherche locale et des métaheuristiques.

Il existe un nombre important de méthodes gloutonnes pour le problème Max Stable (voir par exemple les réferences [Johnson, 1974] et [Tomita et al., 1988]). Le principe de base de ces méthodes consiste à fixer à chaque étape la valeur d'une (ou plusieurs) variable sans remettre en cause les choix effectués précédemment jusqu'à déterminer une solution réalisable. On peut à chaque étape ajouter un sommet à un stable partiel ou supprimer un sommet d'un ensemble non stable. Pour le choix du sommet suivant à traiter et de sa valeur, on fait intervenir des heuristiques. Par exemple, on peut choisir le sommet qui a le plus grand nombre d'adjacents. Ces méthodes sont généralement rapides, mais fournissent le plus souvent des solutions de qualité médiocre.

Contrairement à l'approche gloutonne, la recherche locale manipule des solutions complètes durant la recherche. Fondée sur la notion de voisinage, elle consiste à débuter avec une solution quelconque s, à choisir un voisin $s^{'}$ de s tel que $f(s^{'}) > f(s)$ et à remplacer s par $s^{'}$ puis à répéter ce processus jusqu'à ce que pour tout voisin $s^{'}$ de s, $f(s^{'}) \leq f(s)$. Pour résoudre le problème MC, Feo et al. [Feo et al., 1994] ont proposé une méthode de recherche locale dite méthode de relance aléatoire. Elle consiste à générer une nouvelle configuration de départ de façon aléatoire et à recommencer une descente. Cette étude indique des résultats numériques sur des graphes aléatoires de 1000 sommets et de densité égale à 50%.

Des métaheuristiques ont été aussi proposées pour résoudre les problèmes MS et MC. On cite, par exemple, le recuit simulé, la recherche tabou, les réseaux de neurones et les algorithmes génétiques.

Aarts et Korst [Aarts et al., 1989] ont proposé de résoudre le problème MS par un algorithme de recuit simulé en utilisant une fonction de pénalité pour la fonction coût. Des expérimentations, numériques réalisées sur des graphes aléatoires et des instances de graphes de DIMACS benchmark, ont été présentées dans [Homer et al., 1996]. Les résultats obtenus sont très encourageants.

En 1989, Friden et al. [Friden et al., 1989] ont proposé une approche heuristique, pour le problème MS, basée sur la recherche tabou. Ils ont utilisé 3 listes tabou : une liste pour sauvegarder les dernières solutions visitées et deux listes pour sauvegarder les derniers sommets introduits ou supprimés. Sous certaines conditions, cet algorithme atteint le meilleur voisin en un temps presque constant. Dans [Gendreau et al., 1993, Soriano et al., 1996], trois variantes de recherche tabou pour la résolution du MC sont présentées, où l'espace de recherche consiste à compléter des sous-graphes en maximisant leurs tailles. Les deux premières sont déterministes du fait qu'elles échantillonnent l'espace des voisins, alors que la troisième est probabiliste et échantillonne aléatoirement le voisinage. Ces algorithmes ont donné des résultats très satisfaisants sur des graphes générés aléatoirement. Récemment, Battiti et Protasi [Battiti et al., 1995] ont proposé une extension de la recherche tabou en introduisant une nouvelle méthode de recherche locale. Des tests qui montrent l'efficacité de cet algorithme sont présentés.

Les algorithmes génétiques (AG) ont été aussi utilisés pour résoudre les problèmes MC et

MS. Le premier algorithme génétique a été développé en 1993 par Carter et Park [Carter et al., 1993] pour le problème MC. Pour avoir une bonne performance, plusieurs modifications à l'AG ordinaire ont été effectuées, mais malgré leurs efforts, ils n'ont pas réussi à obtenir des résultats satisfaisants. Plus tard, un travail de Bäck et Khuri [Back et al., 1994] utilisant les AG pour le problème MS a abouti à une conclusion opposée. En utilisant un AG puissant appelé GENEsYs avec une fonction de performance adéquate, incluant un terme de pénalité gradué pour pénaliser les solutions infaisables, ils ont obtenu des résultats intéressants sur des graphes aléatoires et réguliers avec plus de 200 sommets. Depuis, d'autres variantes des AG pour le MC ou le MS ont été proposées. Murthy et al. [Murthy et al., 1994] propose un AG en utilisant un croisement par partie et une mutation modifiée. Ils présentent seulement des résultats sur des graphes de petite taille (jusqu'à 50 sommets), ce qui rend difficile l'évaluation de cet algorithme. Bui et Eppley [Bui et al., 1995] ont également proposé un AG qui a donné des résultats encourageants grâce à une stratégie hybride incorporant une étape de recherche locale à chaque génération. Cette idée a été dévéloppée par Fleurent et Ferland [Fleurent et al., 1996] qui ont utilisé la recherche tabou ainsi que d'autres techniques de recherche locale comme opérateurs de mutation. Les résultats obtenus sont encourageants, néanmoins, le temps de calcul reste élevé. Récemment, Marchiori [Marchiori, 1996] a développé une simple heuristique, basée sur les AG, où il combine un AG et une heuristique gloutonne. Sur les graphes DIMACS benchmark, la performance de cette dernière a dépassé, en terme de qualité de solution et de temps de calcul, tous les AG déjà développés.

De nombreuses approches heuristiques fondées sur les réseaux de neurones ont été aussi proposées pour la résolution du MS et de ses équivalents (voir [Lin et al., 1992], [Jagota et al., 1996], [Grossman, 1996]).

Après ce tour d'horizon des différentes approches développées pour le MS/MC, nous proposons d'étudier l'application de la programmation linéaire en nombres binaires à ce problème. Nous allons voir s'il est ou non intéressant d'utiliser une telle approche, ou plûtot dans quels cas cette approche est intéressante.

4.3 Résolution du problème par la programmation en 0-1

Nous présentons dans cette partie trois formulations en 0-1 et les situons d'un point de vue théorique par rapport à la formulation classique suivante :

$$(MCl)\begin{cases} Max & \sum_{i=1}^{n} x_i \\ s.c. & x_i + x_j \leq 1 \quad (i,j) \in E\,(1) \\ & x_i \in \{0,1\} \quad\;\; i = 1,...,n \end{cases}$$

où n désigne le nombre de sommets du graphe. À chaque arête est associée une contrainte. Elle permet d'assurer que, si un sommet i est selectionné ($x_i = 1$), alors son adjacent j ne le sera pas ($x_j = 0$). Le nombre de contraintes dépend de la densité du graphe et est égal à $|E|$.

4.3.1 Première formulation

Dans cette première formulation, nous essayons de réduire le nombre de contraintes par rapport à la formulation classique. L'idée utilisée est d'agréger verticalement le modèle classique MCl. Soit $d(i)$ le degré du sommet i et $\Gamma(i)$ l'ensemble de ses sommets adjacents. Le modèle obtenu est :

$$(M1)\begin{cases} Max & \sum_{i=1}^{n} x_i \\ s.c. & d(i)x_i + \sum_{j \in \Gamma(i)} x_j \leq d(i) \quad i = 1,...,n \;\; (2) \\ & x_i \in \{0,1\} \qquad\qquad\qquad i = 1,...,n \end{cases}$$

La contrainte (2) n'est autre qu'une aggrégation des contraintes (1). Elle permet d'assurer que, si le sommet i est sélectionné, alors tous ses adjacents ne le seront pas. L'avantage majeur de ce modèle réside dans son petit nombre de contraintes (n contraintes). Cependant, dans la pratique, ce modèle est médiocre vis-à-vis de la qualité de la borne supérieure. En fait, les contraintes du modèle $M1$ sont assez denses, elles contiennent $(d(i) + 1)$ variables, mais ne sont pas toujours actives. Si dans une contrainte i la variable $x_i = 1$, alors toutes les variables x_j associées aux voisins de i prennent la valeur 0. Par contre, si $x_i = 0$, la contrainte i est fortement inactive. Prenons l'exemple suivant : $3x_1 + x_3 + x_4 + x_8 \leq 3$, si $x_1 = 0$ alors la contrainte devient $x_3 + x_4 + x_8 \leq 3$ qui est toujours satisfaite. Pour avoir une borne plus fine, nous proposons de remplacer le coefficient $d(i)$ de la contrainte (2) par une borne supérieure du problème, si cette dernière est plus petite que $d(i)$. Soit UB une borne supérieure du problème MS, on obtient :

$$(M1^{'})\begin{cases} Max & \sum_{i=1}^{n} x_i \\ s.c. & inf(d(i),UB)x_i + \sum_{j\in\Gamma(i)} x_j \le inf(d(i),UB) \quad i=1,...,n \quad (2^{'}) \\ & x_i \in \{0,1\} \qquad\qquad\qquad\qquad\qquad\qquad i=1,...,n \end{cases}$$

Proposition :$(\overline{M1^{'}})$ la relaxation continue de $(M1^{'})$, est plus précise que $(\overline{M1})$, la relaxation continue de $(M1)$.

Preuve : la preuve est évidente.

Pour le calcul de la borne supérieure UB utilisée dans la contrainte (2') du modèle $M1^{'}$, de nombreuses méthodes sont possibles. Nous avons choisi de travailler avec l'une des bornes les plus connues (pas forcément la plus efficace) qui est le nombre chromatique d'un graphe, c'est-à-dire le nombre minimum de couleurs nécessaires pour colorier les sommets d'un graphe sans que deux sommets adjacents soient de la même couleur. En effet, pour un graphe donné G, le nombre chromatique du graphe complémentaire $\overline{G}$ $(\gamma(\overline{G}))$ est une borne supérieure de $\alpha(G)$. Ce dernier problème est également NP-difficile [Garey et al., 1979] et sa résolution de façon exacte est dure. Par contre, différentes méthodes, rapides et faciles à mettre en œuvre, existent pour sa résolution approchée. Nous citons ici trois d'entre elles que l'on désigne par **DSATUR** , **COLOR** et **FRA_COL.**

DSATUR : proposé par Brelaz [Brelaz, 1979], c'est un algorithme séquentiel de coloration appelé methode de Saturation. Soit $DSAT(v)$ le degré de saturation d'un sommet v, défini par le nombre de couleurs différentes utilisées pour les adjacents de v. Si aucun adjacent n'est colorié alors $DSAT(v)$=$degr(v)$. Tant qu'il existe des sommets non coloriés, choisir un sommet v qui maxmise $DSAT$, colorier ce sommet avec la première couleur possible et réitérer.

COLOR : Biggs [Biggs, 1990] propose également un algorithme glouton pour la coloration de sommets. On commence par colorier les sommets non adjacents de degré maximum avec la première couleur. On élimine ensuite du graphe les sommets déjà coloriés et on recommence avec la seconde couleur sur le graphe ainsi réduit et ainsi de suite. Généralement la borne obtenue par l'heuristique COLOR est moins bonne que celle obtenue par DSATUR mais elle nécessite beaucoup moins de temps de calcul.

FRA_COL : la coloration fractionnaire d'un graphe $G = (V, E)$, qui est une relaxation linéaire de la coloration d'un graphe, permet aussi de fournir une borne supérieure pour le nombre chromatique $\gamma(G)$. C'est un ensemble de couleurs fractionnaires $C = \{C_1, C_2, ..., C_k\}$où chaque couleur fractionnaire est définie comme un ensemble stable pondéré par un poids fractionnaire. Une coloration fractionnaire de $G = (V, E)$ est une pondération positive définie sur C telle que pour tout sommet v de V, la somme des poids des couleurs fractionnaires C_i contenant v est supérieure ou égale à 1. La somme totale des poids de l'ensemble des couleurs fractionnaires fournit une borne supérieure du nombre chromatique $\gamma(G)$. Cependant, trouver une coloration fractionnaire avec un poids minimal est NP-difficile. Un bon algorithme itératif approché a été proposé par Balas et Xue [Balas et al., 1996], pour la coloration fractionnaire d'un graphe où à chaque itération la procédure suivante est appliquée. Chaque sommet v de V est ajouté à la première couleur fractionnaire $C_j \in C$, si elle existe, tel que $C_j \cup \{v\}$ est un ensemble stable. Étant donné U l'ensemble des sommets non affectés à une couleur fractionnaire, on cherche une coloration $\left(C_1^i, C_2^i, ..., C_k^i\right)$ de $G(U)$, le graphe induit par U, (en utilisant DSATUR ou COLOR) et on met à jour $C = C \cup \left\{C_1^i, C_2^i, ..., C_k^i\right\}$ et $t_i = |C| / i$. Si $t_i < t_{i-1}$ alors on repète la procédure sinon on arrête l'algorithme. Après i itérations, le poids de chaque couleur fractionnaire sera égal à $1/i$ et chaque sommet sera colorié exactement i fois. Ainsi $t_i = |C| / i$ est une borne supérieure de $\gamma(G)$. La borne obtenue par FRA_COL en utilisant l'heuristique DSATUR, pour l'étape de coloration en chaque itération, est souvent meilleure que celle obtenue par FRA_COL en utilisant l'heuristique COLOR, néanmoins, elle nécessite plus de temps.

On peut également améliorer $M1'$ en remplaçant UB, la borne supérieure du cardinal de tout ensemble stable de G, par UB_i, la borne supérieure du cardinal de tout ensemble stable du sous grahe de G, engendré par les adjacents du sommet i $(\Gamma(i))$. On obtient ainsi le modèle suivant :

$$(M1'') \begin{cases} Max & \sum_{i=1}^{n} x_i \\ s.c. & UB_i x_i + \sum_{j \in \Gamma(i)} x_j \leq UB_i \quad i = 1, ..., n \;\; (2'') \\ & x_i \in \{0,1\} \qquad\qquad\qquad i = 1, ..., n \end{cases}$$

Une autre façon d'améliorer le modèle $M1$ est d'ajouter d'autres inégalités, par exemple des inégalités de cliques, des inégalités triangulaires ou tout autre type de coupe. Ceci fait l'objet du paragraphe suivant.

4.3.2 Deuxième formulation

Pour améliorer la formulation $M1$, nous proposons d'ajouter des inégalités de cliques de type $x_i + \sum_{j \in Cl(i) \setminus i} x_j \leq 1$ avec $Cl(i)$ une clique de sommets de G contenant i. On obtient alors le modèle suivant :

$$(M2) \begin{cases} Max & \sum_{i=1}^{n} x_i & \\ s.c. & d(i)x_i + \sum_{j \in \Gamma(i)} x_j \leq d(i) & i = 1, ..., n \quad (2) \\ & x_i + \sum_{j \in Cl(i) \setminus i} x_j \leq 1 & i = 1, ..., n \quad (3) \\ & x_i \in \{0, 1\} & i = 1, ..., n \end{cases}$$

Les contraintes de type (3) permettent d'assurer que si un sommet i est sélectionné alors tous les autres sommets de $Cl(i)$ ne le seront pas. Même si ces contraintes sont redondantes en variables 0-1, elles permettent d'améliorer la borne supérieure du problème, obtenue par relaxation continue. Le nombre de contraintes de ce modèle est $2n$.

Proposition :$(\overline{M2})$, la relaxation continue de $(M2)$, est plus précise que $(\overline{M1})$, la relaxation continue de $(M1)$.

Preuve : La preuve est évidente.

Afin d'affiner au mieux les contraintes (3), il est intéressant de chercher une clique $Cl(i)$ la plus grande possible. Comme il n'est pas du tout évident de trouver une clique de taille maximale contenant le sommet i, nous avons choisi d'appliquer une simple heuristique. Elle s'énonce comme suit :

Pour tout sommet i de V, $Cl(i) = \{i\}$

1. Pour chaque nœud i, on cherche l'ensemble de ses nœuds adjacents, $\Gamma(i)$,
2. on cherche le nœud $j \in \Gamma(i)$ tel que l'intersection entre $\Gamma(i)$ et l'ensemble des nœuds adjacents à j soit maximale, et on l'ajoute à $Cl(i)$,
3. on supprime de $\Gamma(i)$ les nœuds qui ne sont pas adjacents à j ainsi que j,
4. on repète les étapes 2 et 3 jusqu'à ce que $\Gamma(i)$ devienne vide.

De façon similaire au modèle $M1$, en remplaçant, dans $M2$, les contraintes de type (2) par celles de type (2') (resp. (2")), on obtient le modèle plus précis $M2^{'}$ (resp. $M2^{''}$).

4.3.3 Troisième formulation

Nous avons vu au paragraphe précédent l'avantage des contraintes de type $x_i + \sum_{j \in Cl(i) \backslash i} x_j \leq 1$. C'est pourquoi nous proposons la formulation $(M3)$, dont la relaxation continue fournit une borne supérieure meilleure que celles données par $(M1)$ et $(M2)$. Cette nouvelle formulation est basée aussi sur des inégalités de type clique. Pour chaque arête (i, j), on associe une contrainte dont les termes correspondent aux sommets d'une clique contenant à la fois les sommets i et j. Ce modèle a le même nombre de contraintes que le modèle classique (MCl), par contre ces dernières sont plus denses, ce qui permet d'avoir une borne supérieure meilleure. Soit $Cl(i, j)$, une clique contenant i et j, le modèle $(M3)$ s'ecrit :

$$(M3) \begin{cases} Max & \sum_{i=1}^{n} x_i & \\ S.C. & x_i + x_j + \sum_{k \in Cl(i,j) \backslash i,j} x_k \leq 1 & (i,j) \in E \ (4) \\ & x_i \in \{0, 1\} & i = 1, ..., n. \end{cases}$$

Pour ce modèle également, nous avons choisi d'appliquer une heuristique simple pour déterminer les cliques $Cl(i, j)$. Cette heuristique est décrite ci-dessous :

Pour toute arête $(i, j) \in E$, $Cl(i, j) = \{i,\ j\}$

1. pour chaque arête (i, j), on cherche l'ensemble de sommets adjacents à la fois à i et j $\Gamma(i, j)$,
2. on cherche le sommet $k \in \Gamma(i, j)$ tel que l'intersection entre $\Gamma(i, j)$ et l'ensemble de sommets adjacents à k soit de cardinal maximal, et on l'ajoute à $Cl(i, j)$,
3. on supprime de $\Gamma(i, j)$ les nœuds qui ne sont pas adjacents à k ainsi que k,
4. on repète les étapes 2 et 3 jusqu'à ce que $\Gamma(i, j)$ devienne vide.

Proposition :$(\overline{M3})$ la relaxation continue de $(M3)$, est plus précise que $(\overline{M2})$, la relaxation continue de $(M2)$.

Preuve : L'ensemble des contraintes (3) du modèle $(M2)$ est inclus dans l'ensemble des contraintes (4) du modèle $(M3)$.

4.4 Résultats numériques

Tous les programmes associés aux différents modèles présentés ont éte résolus en utilisant CPLEX 6.0 [Cplex, 1998] sur une machine à processeur alpha à 333 MHz. Les expérimentations ont été réalisées sur des instances de graphes générées aléatoirement et sur des instances de graphes de DIMACS benchmark[1] développées en 1993 pour le challenge DIMACS (voir [Hasselberg et al., 1993]). Ces instances incluent des graphes aléatoires non uniformes (contenant des cliques de grande taille) et des graphes modélisant des problèmes particuliers (coding theory, steiner triple problem, fault diagnosis, vertex cover problems, tilings of hypercubes).

Pour la borne UB (la borne supérieure du cardinal de tout ensemble stable d'un graphe donné), utilisée dans les modèles $M1^{'}$ et $M2^{'}$, nous avons utilisé l'heuristique FRA_COL présentée au paragraphe 4.3.1. L'étape de coloration a été réalisée par l'heuristique COLOR. Elle permet de donner rapidement une borne de bonne qualité.

Pour accélérer la procédure de branch-and-bound, nous avons remarqué qu'il etait préférable de préciser un ordre de branchement, pour les différents modèles proposés, sur les graphes de moyenne et forte densité. En effet, on peut noter que le nombre de variables fixées en chaque nœud du branch-and-bound dépend fortement du branchement. Si le sommet correspondant à la variable de branchement est fixé à 1, alors les variables correspondant à tous ses adjacents seront fixées à zéro. Nous avons testé deux ordres de branchement qui traitent en premier les sommets de grand degré. Le premier ordre consiste à trier les sommets de la façon suivante : on choisit le sommet de degré maximum, on l'élimine du graphe avec toutes ses arêtes et on recommence la même chose sur le graphe réduit. Le deuxième ordre consiste à trier les sommets selon un ordre de coloration. Nous avons choisi d'appliquer l'algorithme glouton de Biggs [Biggs, 1990] pour la coloration des sommets. Rappelons qu'on commence par colorier les sommets non adjacents de degré maximum avec la première 'couleur'. On élimine ensuite du graphe les sommets déjà coloriés et on recommence avec la seconde 'couleur' sur le graphe ainsi réduit, ainsi de suite. Après de nombreuses expérimentations, nous avons retenu le premier ordre de branchement.

Dans le but de réduire l'arbre d'exploration, nous avons fourni au logiciel de programmation une borne inférieure. Cette dernière est obtenue en utilisant une simple heuristique qui consiste à déterminer une clique maximale C pour le graphe $\overline{G} = (V, \overline{E})$ représentant le graphe complémentaire de $G = (V, E)$. Soit $S = V$ et $C = \emptyset$, tant que $S \neq \emptyset$, choisir un sommet $v \in S$ de degré maximum dans $\overline{G}$ et mettre à jour $C = C \cup \{v\}$ et $S = S \cap \Gamma(v)$ où $\Gamma(v)$

[1] `ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique/`

est l'ensemble de sommets adjacents à v dans le graphe $\overline{G}$. Le temps de calcul de cette borne inférieure est négligeable.

Le tableau 4.1 résume les comparaisons numériques entre les trois modèles $M1^{'}$, $M2^{'}$ et $M3$, du point de vue temps de résolution. Les instances considérées concernent des graphes générés aléatoirement de taille $n = 50$, 100 et 200 pour 9 valeurs de densité (de 10% à 90%). Le temps de résolution d'une instance est limité à 5400 secondes. Chaque ligne du tableau présente la moyenne des résultats obtenus en résolution exacte sur 10 problèmes. Pour chaque type de problème, on indique la durée totale de la résolution (Temps) et l'écart relatif entre la borne supérieure obtenue par relaxation continue et la valeur de la solution optimale.

Données		$M1^{'}$		$M2^{'}$		$M3$	
Nb_var	**Densité**	**Écart relatif**	**Temps**	**Écart relatif**	**Temps**	**Écart relatif**	**Temps**
50	10%	19%	0.45	9%	0.12	5%	0.04
	20%	66%	5.43	26%	0.57	20%	0.23
	30%	118%	5.54	72%	0.94	27%	0.28
	40%	110%	4.56	40%	1.36	15%	0.69
	50%	102%	2.43	91%	1.0	23%	1.25
	60%	98%	1.46	84%	1.25	33%	1.49
	70%	81%	0.87	63%	0.98	36%	1.32
	80%	33%	0.46	34%	0.42	28%	1.17
	90%	25%	0.23	25%	0.2	20%	0.67
100	10%	61%	3117.60	32%	110.80	16%	8.56
	20%	145%	2078.35	65%	487.8	30%	78.13
	30%	186%	1727.40	93%	414.2	46%	126.90
	40%	175%	298.76	116%	227.4	58%	106.15
	50%	145%	41.87	133%	36.75	77%	161.03
	60%	134%	17.70	110%	20.43	92.5	179.34
	70%	110%	15.42	90%	13.32	83%	165.80
	80%	60%	9.47	55%	9.18	60%	105.62
	90%	25%	4.85	25%	8.67	40%	66.67
200	50%	273%	4565.3	227%	3963.12	-	-
	60%	233%	1506.38	204%	1126.19	-	-
	70%	162%	392.23	150%	449.65	-	-
	80%	120%	136.65	116%	285.45	-	-
	90%	100%	115.36	100%	192.87	-	-

TAB. 4.1 – Comparaison numérique entre $(M1^{'})$, $(M2^{'})$ et $(M3)$
- : la solution n'est pas obtenue dans la limite de temps de 5400 secondes

Le tableau 4.1 montre que le modèle $M1'$ est le plus efficace pour les fortes densités (de 60% à 90%). Bien que la borne supérieure fournie par la relaxation continue de $M1'$ soit de moins bonne qualité que celles fournies par les relaxations continues de $M2'$ et $M3$, il permet de résoudre des problèmes de grande taille en un temps raisonnable. Par exemple, la durée de résolution d'un problème de 200 sommets et de densité 90% est en moyenne 115 secondes. Par contre, pour les faibles densités, le modèle $M1'$ est le moins efficace (un problème à 100 sommets et de densité égale à 10% nécessite en moyenne 3100 secondes). Pour ces faibles densités, le modèle $M3$ est le plus efficace. Par exemple, le temps nécessaire en moyenne, pour la résolution d'une instance de 100 sommets et de densité 10%, est égal à 8 secondes ce qui est incomparable avec le temps requis par le modèle $M1'$, 3100 secondes. Cependant, son efficacité reste limitée aux problèmes de taille inférieure ou égale à 100. En effet, nous n'avons pu résoudre, dans la limite de temps fixée à 5400 seondes, aucune instance de taille 200, quelle que soit la densité. Par contre, ceci est possible en utilisant les modèles $M1'$ et $M2'$ mais seulement pour des instances à faible et moyenne densité. Le tableau 4.1 montre également que le modèle $M2'$ est le plus efficace pour les moyennes densités, il est un bon compromis entre les modèles $M1'$ et $M3$.

Le tableau 4.2 présente une comparaison entre notre approche et deux autres approches considérées parmi les meilleures approches développées dans la littérature pour le problème MC : l'algorithme de [Balas et al., 1996] et l'algorithme de [Wood, 1997], qui sont de type branch-and-bound et qui utilisent différentes heuristiques de coloration pour la détermination de bornes supérieures. Ces résultats ont été obtenus sur les instances du DIMACS benchmark, développées pour le problème MC. Pour nos expérimentations, nous avons considéré les graphes complémentaires. Pour chaque instance, nous indiquons la taille (Nb_var), la densité du graphe (Densité) et le temps CPU mis par chaque approche (Temps). Notons que $M1' \times M2' \times M3$ désigne le meilleur résultat obtenu par $M1'$, $M2'$ ou $M3$. Les machines utilisées dans [Balas et al., 1996] et [Wood, 1997] sont une machine Sun Sparc 10 et une machine DEC alpha 300-400 respectivement.

Le tableau 4.2 montre que, pour certaines instances, la programmation linéaire en 0-1 est une alternative intéressante. Elle est efficace pour la résolution du MS (respectivement MC) sur des graphes peu (respectivement très) denses. Par exemple, le temps mis pour la résolution de l'instance MANN_a27 est 45 secondes qui est très petit comparé aux temps mis par les deux autres approches (704 secondes et 1362 secondes). Par contre, elle reste médiocre pour certaines instances surtout celles de type brock et de type p_hat.

Données			wood[2]	Balas &Xue[3]	$M1^{'} \times M2^{'} \times M3$
Instance	**Nb_var**	**Densité**	**Temps**	**Temps**	**Temps**
brock200_1	200	75%	805.20	172.19	-
brock200_2	200	50%	3.83	1.90	2400.5
brock200_3	200	61%	38.50	7.96	-
brock200_4	200	66%	92.95	26.00	-
c-fat200-1	200	8%	0.02	0.03	0.01
c-fat200-2	200	16%	0.02	0.02	0.02
c-fat200-5	200	43%	0.13	0.13	0.01
c-fat500-1	500	4%	0.02	0.08	0.09
c-fat500-2	500	7%	0.08	0.11	0.10
c-fat500-5	500	19%	0.17	0.24	0.20
c-fat500-10	500	37%	0.03	0.64	0.65
hammings6-2	64	90%	0.01	0.01	0.01
hammings6-4	64	35%	0.07	0.01	2.1
hammings8-2	256	97%	0.01	0.34	0.01
johnson8-4-4	70	77%	0.02	0.01	0.07
johnson16-2-4	120	76%	195.80	0.01	31.34
keller4	171	65%	18.45	3.39	2342
MANN_a9	45	93%	0.10	0.02	0.03
MANN_a27	378	99%	704.30	1362.82	45.32
p_hat300-1	300	24%	1.47	0.80	-
p_hat300-2	300	49%	10.05	5.83	-
San200_0.7_1	200	70%	0.92	1.69	21.59
San200_0.7_2	200	70%	0.47	4.91	9.53
San200_0.9_1	200	90%	11.48	0.26	0.85
San200_0.9_2	200	90%	1052.00	12.27	2.93

TAB. 4.2 – Comparaison numérique entre notre approche et les approches de [Wood, 1997], [Balas et al., 1996] (graphes de DIMACS benchmark)
- : la solution n'est pas obtenue dans une limte de temps de 5400 secondes

Nous avons constaté également, que la borne obtenue par relaxation continue du modèle $M3$ est souvent plus fine que celles qui sont données par les différentes heuristiques de coloration utilisées dans les algorithmes branch-and-bound de [Balas et al., 1996] et [Wood, 1997] : l'heuristique COLOR, l'heuristique DSATUR, l'heuristique F_COLOR et l'heuristique F_DSATUR. F_COLOR et F_DSATUR sont, respectivement, une coloration fractionnaire approchée en utilisant l'heuristique COLOR et une coloration fractionnaire approchée en utilisant l'heuristique DSATUR. Le tableau 4.3 présente une comparaison entre $\overline{M3}$, la relaxation continue de $M3$, et les 4 heuristiques citées ci-dessus en considérant les instances du DIMACS benchmark. Pour chaque instance, nous indiquons la taille, la densité et la valeur de la borne supérieure obtenue par chacune des méthodes.

Il est clair que la relaxation continue du modèle $M3$ fournit dans la plupart des cas une borne supérieure plus fine. Sur les 27 instances testées, elle donne 25 fois un résultat meilleur. Une amélioration en moyenne de 30% par rapport à **COLOR** la plus mauvaise borne heuristique et de 13% par rapport à **F_DSATUR** la meilleure borne heuristique. Il serait ainsi très intéressant, pour de nombreux algorithmes proposés dans la littérature, d'utiliser la borne obtenue par relaxation du modèle $M3$.

4.5 Conclusion

Nous avons considéré dans ce chapitre, le problème de l'ensemble stable de cardinal maximal d'un graphe. Le but était d'étudier ce que la programmation linéaire en variables binaires peut apporter à la résolution de ce problème. Nous avons proposé trois formulations qui se distinguent en particulier par le nombre et la nature des contraintes mises en jeu. Les résultats expérimentaux montrent que la PLVB est une alternative intéressante pour résoudre ce problème. Elle permet en effet d'obtenir facilement des résultats comparables à des méthodes spécifiques qui demandent une implémentation soigneuse pour être efficace. Néanmoins ce travail reste à poursuivre. Nous envisageons dans de prochains travaux : 1. d'affiner les modèles étudiés dans ce chapitre, en rajoutant de nouvelles inégalités valides ou en raffinant les coefficients de celles déjà utilisées ; 2. d'établir une étude comparative plus précise entre notre approche et les principales approches proposées pour ce problème (une étude comparative sur des instances aléatoires et sur une même machine) ; 3. d'étudier le problème du stable avec des poids. Cette variante ne nécessite pas de modifications majeures par rapport au problème sans poids.

Données			Méthode				
Instance	**Nb_var**	**Densité**	**COLOR**	**DSATUR**	**F_COLOR**	**F_DSATUR**	$\overline{M3}$
brock200_1	200	75%	55	54	49	48	41
brock200_2	200	50%	34	31	28	28	25
brock200_3	200	61%	42	40	36	35	31
brock200_4	200	66%	45	44	39	39	34
c-fat200-1	200	8%	14	15	13	12	12
c-fat200-2	200	16%	28	24	26	24	24
c-fat200-5	200	43%	84	84	74	70	66
c-fat500-1	500	4%	17	14	16	14	14
c-fat500-2	500	7%	32	26	28	26	26
c-fat500-5	500	19%	70	64	69	64	64
c-fat500-10	500	37%	126	126	126	126	126
hammings6-2	64	90%	32	32	32	32	32
hammings6-4	64	35%	8	7	8	7	6
hammings8-4	256	97%	32	24	32	18	18
hammings8-2	256	64%	128	128	128	128	128
johnson8-4-4	70	77%	20	17	20	14	14
johnson16-2-4	120	76%	14	14	14	8	8
keller4	171	65%	28	23	20	18	17
MANN_a9	45	93%	21	19	19	18	18
MANN_a27	378	99%	144	140	137	136	135
p_hat300-1	300	24%	23	21	19	19	24
p_hat300-2	300	49%	46	42	40	39	46
San200_0.7_1	200	70%	40	35	37	35	31
San200_0.7_2	200	70%	18	21	18	21	24
San200_0.9_1	200	90%	76	75	73	71	70
San200_0.9_2	200	90%	78	72	72	70	61
Sanr200_0.7	200	70%	51	45	44	42	36

TAB. 4.3 – Comparaison numérique entre différentes bornes supèrieures (graphes de DIMACS benchmark)

Bibliographie

[Aarts et al., 1989] E. Aarts, J. Korst, 'Simulated annealing and boltzmann machines', J. Wiley & Sons, Chichester, UK (1989).

[Babel, 1991] L. Babel, 'A fast algorithm for the maximum weight clique problem', Computing, vol 46, p. 321-341 (1991).

[Babel et al., 1990] L. Babel, G. Tinhofer, 'A branch and bound algorithm for the maximum clique problem', ZOR-Methods and Models of Operations Research, vol. 34, p. 207-217 (1990).

[Back et al., 1994] T. Bäck, S. Khuri, 'An evolutionary heuristic for the maximum independent set problem', Proc 1st IEEE Conf. Evolutionary Comput., p. 531-534 (1994).

[Balas et al., 1986] E. Balas, C.S. YU, 'Finding the maximum clique in an arbitrary graph', SIAM J. Comput. 15/4, p. 1054-1068 (1986).

[Balas et al., 1996] E. Balas, J. Xue, 'Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring', Algorithmica, vol 15, p. 397-412 (1996).

[Battiti et al., 1995] R. Battiti, M. Protasi, 'Reactive local search for the maximum clique problem', Technical Report TR-95-052, International computer science institute, Berkeley (1995).

[Biggs, 1990] N. Biggs, 'Some Heuristics for graph coloring', in R. Nelson, R.J. Wilson, (Eds), Graph Colourings, Longman, NY, p. 87-96 (1990).

[Bourjolly et al. 1996] J. -M. Bourjolly, P. Gill, G. Laporte, H. Mercure, 'An exact quadratic 0-1 algorithm for the stable set problem', In D.S. Johnson, M.A. Trick (eds), Cliques, Coloring and Satisfiablity : Second DIMACS Implementation Challenge, Dimacs Vol. 26, p. 53-73 (1996).

[Bourjolly et al., 1997] J.-M. Bourjolly, G. Laporte, H. Mercure, 'A combinatorial column generation algorithm for the maximum stable set problem', Oper. Res. Lett. Vol 20, p. 21-29 (1997).

[Brelaz, 1979] D. Brelaz, 'New methods to color the vertices of graph', Commun. ACM, vol. 22, p. 251-256 (1979).

[Bui et al., 1995] T. N. Bui, P.H. Eppley, 'A hybrid genetic algorithm for the maximum clique problem', Proc. 6th Int. Conf. Genetic Algorithms, p. 478-484 (1995).

[Carter et al., 1993] B. Carter, K. Park, 'How good are genetic algorithms at finding large cliques : An experimental study', Technical Report BU-CS-93-015, Computer Science Dept., Boston University (1993).

[Cplex, 1998] Ilpg, Inc., Cplex Division, 'Using the Cplex callable library', Version 6.0 (1998).

[Della et al., 1994] F. Della Croce, R. Tadeai, 'A multi-KP modeling for the maximum clique problem', EJOR, vol 73, p. 555-561 (1994).

[Ebenegger et al., 1984] C.Ebenegger, P.L. Hammer, D. de Werra, "Pseudo-boollean functions and stability of graphs", Ann. Discr. Math., vol 19, p. 83-98 (1984).

[Feo et al., 1994] T.A. Feo, M.G.C. Resende, S.H. Smith, 'A greedy randomized adaptive search procedure for maximum independent set', Oper. Res., Vol. 42, p. 860-878 (1994).

[Fleurent et al., 1996] C. Fleurent, J.A. Ferland, 'Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability', In D.S. Johnson, M.A. Trick (eds), Cliques, Coloring and Satisfiablity : Second DIMACS Implementation Challenge, Dimacs Vol. 26, p. 619-652 (1996).

[Friden et al., 1989] C. Friden, A. Hertz, M. de Werra, 'STABULUS : A technique for finding stable sets in large graphs with tabu search', Computing, Vol. 42, p. 35-44 (1989).

[Friden et al., 1990] C. Friden, A. Hertz, M. de Werra, 'TABARIS : An exact algorithm based on tabu search for finding a maximum independent set in a graph', Comput. Oper. Res., vol 17, p. 437-445 (1990).

[Homer et al., 1996] S. Homer, M. Peinado, 'Experiments with polynomial-time clique approximation algorithms on very large graphs', In D.S. Johnson, M.A. Trick (eds), Cliques, Coloring and Satisfiablity : Second DIMACS Implementation Challenge, Dimacs Vol. 26, p. 147-167 (1996).

[Gendreau et al. 1988] M. Gendreau, J. -C. Picard and L. Zubieta, "An efficient implicit enumeration algorithm for the maximum clique problem", Springer-Verlag, Lecture Notes in Economics Mathematical Systems 304, A. Kurzhanski et al. (eds.), p. 79-91 (1988).

[Gendreau et al., 1993] M. Gendreau, L. Salvail, P. Soriano, 'Solving the maximum clique problem using a tabu search approach', Ann. Oper. Res., Vol. 41, p. 385-403 (1993).

[Grossman, 1996] T. Grossman, 'Applying the INN model to the max clique problem', In D.S. Johnson, M.A. Trick (eds), Cliques, Coloring and Satisfiablity : Second DIMACS Implementation Challenge, Dimacs Vol. 26, p. 125-146 (1996).

[Hasselberg et al., 1993] J.Hasselberg, P.M. Pardalos, G. Vairaktarakis, 'Test case generators and computational results for the maximum clique problem', J. Global Optimasation, Vol. 3, p. 463-482 (1993).

[Jagota et al., 1996] A. Jagota, L. Sanchis, R. Ganesan, 'Approximately solving maximum clique using neural networks and related heuristics', In D.S. Johnson, M.A. Trick (eds), Cliques, Coloring and Satisfiablity : Second DIMACS Implementation Challenge, Dimacs Vol. 26, p. 169-204 (1996).

[Johnson, 1974] D.S. Johnson, 'Approximation algorithms for combinatorial problems', J. Comput. Syst. Sci., vol 9, p. 256-278 (1974).

[Kikustus, 1986] P. Kikustus, 'Another algorithm determing the independance number of a graph', Elektron. Inf. Verarb Kybern. ELK 22, p. 157-166 (1986).

[Lin et al., 1992] F. Lin, K. Lee, 'A parallel computation network for the maximum clique problem' In Proceeding 1st Int. Conf. Fuzzy Theory Techniques, Baton Rouge, Louisiana (1992).

[Loukakis et al., 1982] E. Loukakis and C. Tsouros, 'Determining the number of internal stability of a graph', Internat. J. Comput. Math. 11, p. 232-248 (1982).

[Carraghan et al., 1990] R. Carraghan and P.M. Pardalos 'An exact algorithm for the maximum clique problem', Operations Resaerch Letters 9, p. 375-382 (1990).

[Garey et al., 1979] M. Garey and D. Johson, 'Computers and Intractability- A guide to the theory of NP-Completeness', Freeman, New York, 1979.

[Marchiori, 1996] E. Marchiori, 'A simple heuristic based genetic algorithm for the maximum clique problem', Proc. ACM Symp. Appl. Comput., p. 366-373 (1998).

[Murthy et al., 1994] A.S. Murthy, G. Parthasarathy, V.U.K. Sastry, 'Clique finding-A genetic approach', Proc. 1st IEEE Conf. Evolutionary Comput., p. 18-21 (1994).

[Ostergard, 2002] P.R.J. Östergard, 'A fast algorithm for maximum clique problem', Disc. App. Math., vol 120, p. 19-207 (2002).

[Pardalos et al., 1990] P.M. Pardalos, A.T. Phillips, 'A global optimization approach for solving the maximum clique problem', Int. J. Comput. Math., vol 33, p. 209-216 (1990).

[Pardalos et al., 1992] P.M. Pardalos, G.P. Rodegers, 'A branch and bound algorithm for the maximum clique problem', Comput. Oper. Res., vol 19, p. 363-375 (1992).

[Robson, 1986] J.M. Robson, 'Algorithms for maximum independent sets', J. Algorithms, vol 7, p. 425-440 (1986).

[Sewell, 1998] E.C. Sewell, 'A branch and bound algorithm for the stability number of a sparse graph', INFORMS J. Comput, vol 10, p. 438-447 (1998).

[Soriano et al., 1996] P. Soriano, M. Gendreau, 'Diversification strategies in tabu search algorithms for the maximum clique problem,' Ann. Oper. Res., Vol. 63, p. 189-207 (1996).

[Tarjan et al., 1977] R.E. Tarjan and A.E. Trojanowski, 'Finding a maximum independent set', SIAM J. Comput. 6/3, p. 537-546 (1977).

[Tomita et al., 1988] E. Tomita, S. Mitsuma, H. Takahashi, 'Two algorithms for finding a near-maximum clique', Technical report UEC-TR-C1 (1988).

[Wood, 1997] D.R. Wood, 'An algorithm for finding a maximum clique in a graph', Oper. Res. Lett, vol 21, pp 211-217, 1997.

[Xue, 1991] J. Xue, 'Fast algorithms for vertex packing and related problems', Ph.D Thesis, GSIA, Carnegie Mellon University (1991).

Conclusion générale

Le chercheur opérationnel confronté à des problèmes d'optimisation difficile peut essayer de les aborder des deux façons suivantes : 1. Formulation du problème, modélisation, conception et implémentation d'un algorithme spécifique. Chaque étape est très importante et doit être réalisée avec beaucoup de soin. Les tâches de conception et d'implémentation sont souvent très lourdes et demandent couramment plusieurs mois de travail. 2. Formulation du problème et résolution par un solveur. Dans ce cas les tâches d'implémentation sont beaucoup moins lourdes. En revanche la modélisation du problème doit être la plus fine possible. Le travail de modélisation peut être assez délicat car il existe souvent de nombreuses façons de modéliser (au sens large) un même problème et le choix d'une modélisation peut permettre de résoudre le problème ou au contraire conduire à un échec. Il est intéressant d'effectuer en priorité la deuxième approche. Il peut être également très efficace de combiner deux approches : l'algorithme retenu est alors un algorithme spécifique qui utilise un solveur standard pour la résolution d'un certain nombre de sous-problèmes.

Il existe aujourd'hui de nombreux logiciels extrêmement performants pour la résolution de certains programmes mathématiques en variables mixtes (PMVM). On peut ainsi résoudre de grandes instances de programmes *linéaires* en variables mixtes et de programmes *quadratiques* en variables mixtes à condition, dans ce dernier cas, que la fonction économique soit convexe et que les contraintes soient linéaires. L'objectif principal de ma thèse était de montrer que la résolution pratique de nombreux problèmes d'optimisation combinatoire difficiles pouvait se faire à travers ce modèle général de la programmation mathématique en variables mixtes.

Pour de nombreux problèmes d'optimisation combinatoire les meilleures méthodes proposées sont généralement très spécifiques - une petite variation dans l'énoncé du problème peut demander une refonte importante de l'algorithme- et souvent difficiles à mettre en oeuvre. Nous avons considéré dans ce travail quatre problèmes très différents : un problème de cheminement optimal dans un graphe, un problème de gestion optimale de portefeuille, un problème d'optimisation fractionnaire en variables bivalentes et un problème de détermination d'un ensemble remarquable de sommets dans un graphe ; nous avons étudié leur résolution par la PMVM.

Au premier chapitre, nous avons étudié la résolution d'un problème de plus court chemin avec incertitude sur les données, plus précisément, le problème du 'plus court chemin robuste absolu'. Nous avons montré que, contrairement aux méthodes proposées dans la littérature, la programmation linéaire en variables mixtes permettait de résoudre facilement et de façon efficace de grandes instances du problème.

Le deuxième chapitre est consacré au problème de gestion optimale d'un portefeuille. Nous avons traité deux variantes, importantes en pratique, de ce problème : la construction de la frontière d'efficience d'un portefeuille avec une contrainte sur le nombre de titres à sélectionner et le rééquilibrage d'un portefeuille avec des coûts de transactions comportant une partie fixe et une partie linéaire. Nous avons formulé ces problèmes par des programmes quadratiques en variables mixtes et avons proposé des méthodes heuristiques fondées sur la relaxation continue du programme, couplée avec une procédure branch-and-bound tronquée. Les résultats expérimentaux numériques obtenus sur les données de cinq marchés mondiaux sont particulièrement bons par rapport à d'autres méthodes proposées dans la littérature.

Nous avons considéré au troisième chapitre un problème d'optimisation combinatoire fractionnaire, la maximisation de la somme de plusieurs ratios hyperboliques en variables bivalentes. Nous avons proposé plusieurs types de linéarisations pour ce problème. Ces différentes linéarisations se distinguent par les expressions linéarisées et les contraintes et variables additionnelles nécessaires. Une constatation importante est que les performances de ces différentes linéarisations dépendent fortement des coefficients des contraintes (de linéarisation), ces coefficients n'étant pas définis de façon unique par la linéarisation choisie. Nous avons également proposé différentes stratégies de résolution et notamment un algorithme branch-and-bound spécifique, fondé sur une réécriture plus précise des programmes correspondant à la linéarisation du problème en chaque nœud de l'arbre de recherche. Ainsi en chaque nœud de l'arbre on essaye d'améliorer la linéarisation du problème associé à ce nœud en jouant sur les 'coefficients de linéarisation' ; on ne se contente pas de la formulation linéaire initiale (à la racine de l'arbre). Les résultats expérimentaux que nous avons obtenus fournissent un point de comparaison intéressant pour tester l'efficacité de nouveaux algorithmes puisque, à notre connaissance, aucun résultat expérimental n'a été publié sur ce problème fractionnaire réputé difficile.

Enfin, dans le quatrième chapitre, nous avons traité le problème de l'ensemble stable de cardinal maximal d'un graphe. Nous avons proposé différentes formulations par la programmation linéaire en variables binaires. Les résultats expérimentaux montrent que cette approche est une alternative intéressante pour résoudre ce problème qui a été très largement étudié. Elle permet en effet d'obtenir facilement des résultats comparables à des méthodes spécifiques, souvent très élaborées, qui demandent une implémentation soigneuse pour être efficace.

La conclusion de ce travail est triple : 1. Il existe généralement plusieurs modélisations d'un même problème d'optimisation combinatoire par un programme linéaires en variables mixtes (PLVM) et, pour un type de modélisation retenu, l'efficacité du PMVM considéré dépendra fortement du choix de certains coefficients. En effet certains coefficients du programme proviennent uniquement de la transformation du problème initial en un PMVM et ne sont pas déterminés de façon unique. Ces coefficients peuvent, par exemple être déterminés dans une phase de pré-traitement qui est particulièrement importante pour la résolution pratique de ces problèmes. 2. Pour les quatre problèmes que nous avons étudiés la PMVM est aussi efficace ou même plus efficace dans certains cas que les meilleurs algorithmes qui ont été publiés. D'un point de vue efficacité, l'approche est donc déjà très intéressante. Elle l'est également pour des raisons de fiabilité, de facilité d'implémentation et d'adaptabilité à de petites variations du problème. 3. Les idées utilisées pour les quatre problèmes cités plus haut peuvent être réutilisées de façon directe pour la résolution de beaucoup d'autres problèmes.