Projet recherche 2014
Compte-rendu 1
Appropriation du sujet et premiers éléments de résolution

Adrien EHRHARDT

28 janvier 2014

Table des matières

Première partie

Introduction

Chapitre 1

A quoi sert ce document ?

Dans le cadre du projet recherche, démarche que je détaille dans le chapitre suivant, j'ai entrepris dès le début des créneaux attribués de faire des rapports. Ces rapports serviront à la fois à coucher par écrit les connaissances que j'ai acquises de par mes lectures et mes essais "à la main" ainsi qu'à présenter mes résultats lorsqu'il s'agira de passer à la pratique ; à terme ces rapports sont donc aussi destinés à mon tuteur pour vérifier l'avancée de mon travail, ma compréhension du sujet et mes résultats mais aussi aux personnes externes susceptibles de lire ces rapports avant la soutenance orale de fin de projet recherche.

Comme dit précédemment, la chapitre suivant s'intéresse à la démarche du projet recherche tandis que le suivant le replace dans le contexte de mon projet professionnel.

La partie suivante s'intéresse à la programmation linéaire ; ce n'est pas le coeur du sujet de ce projet recherche mais cette présentation est nécessaire pour préciser le contexte à la fois de l'optimisation mais aussi de la finance et ainsi présenter quelques exemples et résultats. J'y introduis aussi la programmation linéaire entière à travers un exemple concret, tandis que les algorithmes à mettre en place ne seront présentés que dans la partie suivante. La dernière partie est consacrée au sujet principal : la construction d'un indice, la démarche, les critères, les algorithmes disponibles et les résultats préliminaires.

Bonne lecture !

Chapitre 2

Projet Recherche

J'ai souhaité effectuer un projet recherche pour plusieurs raisons. La première est que c'était pour moi l'occasion de mener un travail indépendant, aux heures qui me conviendraient le mieux, et de le faire comme je l'entendais. Cela implique par exemple d'apprendre LaTeX. Ensuite il me fallait trouver un sujet susceptible de m'intéresser au point d'y passer deux jours par semaine jusqu'à la fin de l'année et dont l'objectif final me satisferait. C'est une bonne occasion d'apprendre des choses complexes par soi-même. C'est donc naturellement que, souhaitant faire carrière en finance de marché, et plus particulièrement tout ce qui touche aux mathématiques financières, je me suis tourné vers l'optimisation. Cette approche pratique de la finance par le minimisation ou la maximisation d'un objectif m'a séduit et les applications sont nombreuses.

Il est prévu que le projet recherche s'organise de cette manière :

— Assimilation de la littérature existante sur l'optimisation financière et plus particulièrement la construction d'un indice ;
— Réalisations d'exercices disponibles dans la littérature pour assurer la compréhension de la théorie ;
— Choix de méthodes à implémenter sur des logiciels d'optimisation (à clarifier avec le tuteur)
— Tests des méthodes d'optimisation sur des jeux d'exemples pertinents
— Comparaison des méthodes, conclusions sur l'investissement dans un indice et sa construction
— Rédaction du rapport et de la soutenance

Chapitre 3

Projet professionnel

Candidatant au double-diplôme en Ingénierie Financière à l'Ecole Polytechnique Fédérale de Lausanne, programme de Master en 2 ans, je souhaite m'orienter vers la finance de marché, et notamment ses mathématiques financières. Ce projet recherche est donc une approche indirecte de ces métiers et me permet d'apprendre à la fois des langages de programmation en optimisation ainsi que LaTeX, très utilisé à l'EPFL. Ensuite, d'un point de vue purement pédagogique, les outils qu'il me manque pour l'instant concernant les métiers de la finance ne sont pas les maths (j'ai poursuivi une licence 3 de mathématiques en parallèle de Centrale) ou l'informatique (bien que ces domaines restent à être travaillés spécifiquement en finance) mais les connaissances liées à la finance et ses rouages, ses produits, ses termes techniques... Ce sont des choses que je suis amené à apprendre à travers ce projet recherche que je serai amené à valoriser dans le futur.

Deuxième partie

Programmation linéaire

Chapitre 4

Introduction

En cours de G2 d'Analyse Numérique et Optimisation, nous sommes amenés à traiter les cas de programmation linéaire et partiellement les cas de programmation non linéaire.

Bien que ce ne soit explicitement pas le sujet du projet recherche et que ce soit la partie "facile" de l'optimisation, je vais donner ici quelques exemples de problèmes d'optimisation linéaire en finance (qui hormis qu'ils soient liés à la finance sont strictement équivalents à ce que l'on a pu voir en cours de G2). On se basera sur deux exemples illustrés dans *Optimization Methods in Finance*.

Premièrement, un programme linéaire signifie que la fonction objectif à maximiser ou minimiser est linéaire, de même que les contraintes qui lui sont liées.

Exemple Prenons l'exemple d'une entreprise devant financer un projet avec les Cash-Flow suivant :

Mois	Janvier	Février	Mars	Avril	Mai	Juin
Cash-Flow	-150	-100	200	-200	50	300

Fonction objectif Puisqu'il s'agit de finance, l'objectif est de maximiser le bénéfice ...On appelle v cette variable, qui se retrouvera dans les contraintes.

Variables de décision Ici les moyens de financement de l'entreprise sont au nombre de trois :

- Un crédit bancaire de maximum 100$ à 1% par mois
- L'entreprise peut émettre des obligations de 90 jours à 2% (pour les 3 mois)
- L'excédent peut être placé en banque à 0,3% par mois

Nous choisissons les variables de décision suivantes où l'indice i désigne le mois (1 correspondant à janvier, 6 à juin) :

- x_i : le montant emprunté à la banque
- y_i : le montant issu de la vente des obligations
- z_i : l'argent en excès

Contraintes Ecrivons la contrainte pour le mois de janvier. Nous pouvons emprunter x_1 à la banque, émettre y_1 obligations et garder pour nous z_1 en excès pour combler un cash-flow négatif de 150$ d'où la contrainte : $x_1 + y_1 - z_1 = 150$

En février, nous recevons des intérêts sur l'argent stocké en janvier, i.e. z_1 mais devons rembourser y_1 avec des intérêts d'où la contrainte : $x_2 + y_2 - 1{,}01x_1 + 1{,}003z_1 - z_2 = 100$

Notons enfin qu'à partir d'avril nous ne pouvons plus émettre d'obligations. En revanche nous devons rembourser les obligations arrivés à maturité.

Problème linéaire complet

$$
\begin{array}{llllllllllll}
\max & v & & & & & & & & & & \\
 & x_1 & + & y_1 & - & z_1 & = & 150 & & & & \\
 & x_2 & + & y_2 & - & 1{,}01x_1 & + & 1{,}003z_1 & - & z_2 & = & 100 \\
 & x_3 & + & y_3 & - & 1{,}01x_2 & + & 1{,}003z_2 & - & z_3 & = & -200 \\
 & x_4 & - & 1{,}02y_1 & - & 1{,}01x_3 & + & 1{,}003z_3 & - & z_4 & = & 200 \\
 & x_5 & - & 1{,}02y_2 & - & 1{,}01x_4 & + & 1{,}003z_4 & - & z_5 & = & -50 \\
 & x_6 & - & 1{,}02y_3 & - & 1{,}01x_5 & + & 1{,}003z_5 & - & v & = & -300 \\
 & x_1 & \leq & 100 & & & & & & & & \\
 & x_2 & \leq & 100 & & & & & & & & \\
 & x_3 & \leq & 100 & & & & & & & & \\
 & x_4 & \leq & 100 & & & & & & & & \\
 & x_5 & \leq & 100 & & & & & & & & \\
 & x_i,\, y_i,\, z_i & \geq & 0 & & & & & & & &
\end{array}
$$

Résultats Pour résoudre ce genre de modèles, on utilise des algorithmes connus et déjà présents dans les logiciels de résolution de problèmes linéaires. On peut penser notamment à la méthode du simplex vue en cours d'ANO. Ce programme linéaire peut être résolu grâce au solveur d'Excel comme on a pu le faire en TP d'ANO, ou avec d'autres logiciels spécialisés.

Ici, en utilisant Excel et son solveur, on trouve que le profit maximal réalisé par l'entreprise est 92,50$. Pour ce faire, en janvier 150 $ d'obligations sont émises. En février, 49 $ d'obligations sont émises et 51$ seront empruntés à la banque. En mars, ce sont 203,4 $ d'obligations qui sont émises, ce qui fait un excès de liquidités de 352$ environ. Cet argent est placé et dépensé en avril.

Si on additionne les cash-flow et qu'on ne tient pas compte du coût de l'argent, on peut dire que l'entreprise a réussi à financer son projet pour 7,5$ sur 6 mois en optimisant les moyens de financement !

Au-delà de l'aspect terre-à-terre de la décision Go/No Go au début du projet pour évaluer le profit finalement réalisé par l'entreprise, un modèle comme celui-ci, une fois implémenté, fournit de précieuses informations comme la sensibilité des variables : si au cours du projet on s'aperçoit que les données du problème changent, il suffit de regarder le modèle pour savoir si le plan de financement doit être changé ou non ainsi que l'impact de ces changements sur le profit à venir. Cette analyse est très développée au point 3.3.1 de *Optimization Methods in Finance*.

C'est donc à travers cet exemple basique que l'on retrouve la force de l'optimisation en finance : c'est un outil de prise de décision avant tout et auquel on se réfère constamment lorsqu'on a plus d'informations sur les contraintes de notre problème. En aucun cas le modèle n'est simplement informatif !

Chapitre 5

Programmation linéaire entière

Dans cette partie, je vais continuer en donnant un exemple de problème d'optimisation plus proche que le précédent de ce que je serai effectivement amené à faire pour construire un indice. Les algorithmes détaillés à utiliser pour résoudre ces problèmes, plus complexes que ceux purement linéaires, font l'objet d'une partie dédiée (la prochaine en fait) que l'on peut dans un premier temps passer pour se concentrer sur la Construction d'un index, quitte à y revenir lorsque je parlerai dans cette partie des algorithmes plus spécifiquement.

Une nécessité logique Le profit, et bien d'autres variables de décision que nous avons pu manipuler en ANO, sont des réels sans autre contrainte. Or on comprend bien que si on parle en nombre de patates, en hommes, ...ou en actions, on ne peut pas en avoir 5,2146 ...C'est pourquoi on doit parfois se restreindre à des entiers.

Comme la contrainte de positivité des variables de décision (qui venait elle aussi logiquement de la constatation qu'on ne peut pas emprunter d'argent "négativement" ...), la contrainte "tel variable de décision est entière" s'ajoute aux contraintes existantes du problème d'optimisation. Avec les mains, on comprend bien que le problème résultant ne peut être linéaire, tout comme la fonction "valeur entière" n'est pas continue.

On sera amené plus tard à parler de modèles mixtes : ce sont des modèles dont certaines variables sont entières et d'autres non. A l'inverse, nous avons d'une part les problèmes linéaires vus en section précédente et les problèmes entiers où toutes les variables sont entières. On sera également amené à parler des problèmes binaires : ce sont des problèmes entiers dont les variables ne peuvent prendre que deux valeurs : 0 ou 1. L'état de la recherche actuellement en optimisation consiste essentiellement en l'étude des problèmes non linéaires mixtes (où certaines contraintes sont entières).

Un exemple simple Concrètement les modèles linéaires peuvent servir à modéliser des coûts fixes ou des investissements qui nécessitent un montant précis comme dans l'exemple suivant : Un particulier dispose de 19 000 $ et on lui propose des investissements de 6 700 $, 10 000 $, 5 500 $ et 3 400 $ avec des NPV respectifs de 8 000 $, 11 000 $, 6 000 $ et 4 000 $. La logique (celle d'un problème purement linéaire) voudrait que l'on prenne uniquement l'investissement avec le plus fort Rate of Return, i.e. max (coût / NPV) et en prendre proportionnellement au montant total que l'on peut investir. Ici on comprend bien que cela n'est pas possible, puisque cela reviendrait à réaliser l'investissement 1 2,83 ...fois ce qui n'est pas possible. Alors pourquoi ne pas arrondir ce nombre à deux et réitérer le procédé sur la deuxième valeur max ? On verra

que tel quel, ce n'est pas possible mais qu'un algorithme de résolution se base sur une réflexion similaire.

Pourquoi ce n'est pas tout simplement la valeur entière ? Dans le problème précédent, on voit bien que la solution ne peut pas être la valeur entière de toutes les variables de décision, puisque cela reviendrait à effectuer 2 fois le premier investissement et aucun autre investissement. Il nous resterait alors 5 600 $ que nous pourrions investir dans l'investissement 3 ou 4 ...Il apparaît donc que ce n'est pas la bonne solution. De plus, le problème original est un problème binaire, on ne peut réaliser qu'une fois l'investissement 1 ! On pourra également penser à l'exemple donné en TP d'ANO : on dispose de lingots de métaux de compositions fixes en autres métaux. Nous souhaitons obtenir un produit final avec des spécifications restreintes en composition. On comprend bien qu'un lingot supplémentaire d'un métal fortement concentré en un composant fera exploser la concentration finale du produit, alors que l'on dispose de lingots dont la composition en cet élément est plus restreinte...

La solution Le problème d'investissement cité plus haut a pour solution exactement le contraire de ce que l'on aurait pu s'imaginer : l'investissement 1 n'est en fait pas réalisé, même si son taux de rentabilité est le meilleur. La solution est x_1=0, x_2=1, x_3=1 et x_4=1 où x_i représente la variable de décision binaire d'investir dans i (x_i=1) ou pas (x_i=0).

Troisième partie

Programmation mixte linéaire/entière

Chapitre 6

Introduction

On a vu dans le chapitre précédent à travers un exemple concret de problème d'optimisation appliqué à la finance que les résolutions linéaires en nombres entiers peuvent se révéler compliquées. En effet, il ne s'agit pas d'"arrondir" la solution linéaire ...Il faut donc des algorithmes spécifiques aux problèmes en nombres entiers. C'est exactement ce que l'on se propose de faire dans cette partie : on va exposer différents algorithmes définis dans la littérature et déjà éprouvés : Branch and Bound ainsi que Cutting Planes. On s'intéressera ensuite aux travaux plus récents et notamment Branch and Cut combinant les deux algorithmes précédents. On remarquera que les deux premiers algorithmes datent des années 60 et que depuis, peu d'autres résultats ont vu le jour, à l'exception de la combinaison de ces deux algorithmes.

On s'intéressera dans les chapitres suivants aux heuristiques. Ces algorithmes permettent de trouver facilement (temps polynomial) une solution réalisable sans toutefois garantir qu'il s'agit de la solution optimale. Cela permet notamment d'accélerer les algorithmes de résolution précités. On parlera notamment de Feasibility Pump, une heuristique spécialement conçue pour l'algorithme Branch and Cut.

De la nécessité d'un algorithme Souvenons-nous des débuts de l'ANO ...On se limitait alors (cf les premiers TD) à des problèmes 2D c'est-à-dire ayant seulement deux variables de décision. La région admissible peut alors être représentée par un polytope et on sait que la solution optimale est un point extrême. Il suffisait alors de calculer la fonction objectif à tous ces points extrêmes, trouver le maximum (resp. minimum) de la fonction objectif et ainsi obtenir la solution optimale (x_1,x_2).

En suivant le même raisonnement (toujours en 2D) pour l'optimisation en nombres entiers, on peut affirmer que dans la région admissible (le polytope), il y a un nombre finis de couples de nombres entiers. On peut alors évaluer la fonction objectif en chacun de ces points et procéder comme en programmation linéaire. Or on comprend bien que le nombre des ces points augmente de manière exponentielle avec la taille du problème ! Cette solution est donc vite rendue impossible par le temps de calcul en ces nombreux points.

Dans toute la suite de cette partie, on considère le problème mixte (1) suivant :

$$\begin{array}{lccc} z_I & = & \min & c^T x \\ & Ax & \geq & b \\ & x & \geq & 0 \\ & x_j \text{ entier} & \text{pour} & \text{j} = 1, \ldots, \text{p} \end{array}$$

où x et $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ et $A \in \mathbb{R}^m \times \mathbb{R}^n$. Quitte à renuméroter, on a posé E = 1,...,p où 1 $\leq$ p $\leq$ n représente les variables de décision entières et C = p+1,...,n les variables de décision

réelles.

Problème relaxé Dans le suite du document, on parlera souvent de "problème relaxé" : il s'agit du problème linéaire en nombres entiers avec $E = \varnothing$ (en quelque sorte le problème linéaire engendré par (1)). On verra que dans tous les algorithmes de résolution des MILP (l'acronyme anglais des problèmes linéaires en nombres entiers mixtes - rien que ça), on se ramène toujours au cas purement linéaire à l'aide de ces "problèmes relaxés".

Chapitre 7

Branch and bound - Séparation et évaluation

L'algorithme branch and bound construit un arbre de problèmes relaxés. On part du problème relaxé de (1) et on ajoute des contraintes aux variables de décision qui, dans la solution optimale, doivent être entière comme $x_j \leq u_j$ (noeud fils gauche) ou $x_j \geq l_j$ (noeud fils droit) où l_j - u_j = 1 - on détaillera plus loin les raisons de ce choix.

7.1 Algorithme de base

On s'intéressera à 3 variables dans la boucle de l'algorithme : le noeud courant que l'on résout (N_i), la valeur de la fonction objectif à ce noeud (z_i), ainsi que le vecteur solution (x_i). Nous comparerons ces dernières valeurs à z_U et x^* qui sont actualisés dans certains cas et représentent respectivement un majorant de la fonction objectif ainsi que le vecteur pour lequel ce majorant est atteint. Concernant la structure de données à utiliser, on peut considérer que les noeuds restants à tester sont contenus dans une liste L.

1. Initialisation
   L=N_0 ; N_0 = MILP relaxé ; z_0=z_U=$+\infty$; x_0=x^*=0
2. Condition d'arrêt
   La solution x^* est optimale dès lors qu'il n'y a plus aucun noeud à tester...i.e. dès que L=$\varnothing$
3. Sélection
   On choisit un noeud N_i de L (la méthode de sélection sera discutée plus tard) et on le supprime des noeuds restants à tester contenus dans L.
4. Bound - Evaluation
   On résout N_i (rappel : c'est un problème linéaire). Si le problème est infaisable, on revient en 2., sinon on stocke z_i et x_i.
5. Prune - Taille
   C'est dans cette partie de l'algorithme que nous allons litérallement tailler l'arbre : réduire le nombre de noeuds à tester.
   Si $z_i \geq z_U$, on revient en 2. : c'est que le noeud courant et tout ses noeuds fils auront une fonction objectif supérieure à z^*, la meilleure solution optimale entière (i.e. faisable pour MILP) trouvée jusqu'à présent.

Si x_i n'est pas faisable pour (MILP) (i.e. certaines variables devant être entières ne le sont pas mais la fonction objectif est suffisament grande), on va à l'étape 6. pour explorer l'arbre plus en profondeur.

Si x_i est faisable pour (MILP) (i.e. toutes les variables devant être entières dans le problème original le sont ici ET la fonction objectif est meilleure que la meilleure solution de (MILP) trouvée jusqu'à présent), alors on actualise les valeurs optimales : $z_i{=}z_U$ et $x_i{=}x^*$; on peut alors réactualiser l'arbre avec le premier critère de 5. : on supprime tous les noeuds de L qui vérifient $z_i{\geq}z_U$ et on va en 2.

6. Branch - Séparation

   La méthode de séparation sera l'un des principaux sujets de discussion dans ce qui va suivre, on peut néanmoins donner la structure générale : à partir du noeud courant N_i on construit des sous-problème $N_i^1 \ldots N_i^k$ avec S_p ($1{\leq}p{\leq}k$) la région admissible de N_i^p plus "petite" que S la région admissible de N_i. Tout cela sera quantifié plus tard ! Ces noeuds sont ajoutés à L et on recommence en 2.

Nota Bene : Si la fonction objectif est à maximiser le signe $\geq$ est à changer au profit de $\leq$ et $z_0{=}z_U{=}\text{-}\infty$

Avant de rentrer dans le détail des fonctions de sélection et de séparation pour optimiser l'algorithme, je vais l'illustrer sur deux cas simples.

7.2 Exemple 1

L'algorithme précédent semble bien mystérieux puisque de nombreuses choses sont restées en suspend, et en particulier, le plus important selon moi, la manière de se ramener à des problèmes purement linéaires.

La méthode de sélection des noeuds est un parcours préfixe de l'arbre binaire. La méthode de séparation est la suivante : si par exemple dans la solution x_i^* de N_i, x_i^k* n'est pas entier et devrait l'être

— on ajoute en noeud fils gauche (N_i^1) le même problème linéaire qu'en N_i en ajoutant la contrainte $x_i^k \leq \lfloor\, x_i^k* \,\rfloor$

— on ajoute en noeud fils droit (N_i^2) le même problème linéaire qu'en N_i en ajoutant la contrainte $x_i^k \geq \lceil\, x_i^k* \,\rceil$

Nous allons illustrer cette méthode sur l'exemple trivial suivant :

$$\begin{aligned} \max \quad & x_1{+}x_2 \\ & \text{-}x_1 + x_2 \leq 2 \\ & 8x_1 + 2x_2 \leq 19 \\ & x_1,\, x_2 \geq 0 \\ & x_1,\, x_2 \text{ entiers} \end{aligned}$$

On va à l'étape 1. : N_0 est donc le problème linéaire relaxé suivant :

$$\begin{aligned} \max \quad & x_1{+}x_2 \\ & \text{-}x_1 + x_2 \leq 2 \\ & 8x_1 + 2x_2 \leq 19 \\ & x_1,\, x_2 \geq 0 \end{aligned}$$

Problème que nous savons résoudre facilement, par exemple à l'aide du solveur d'Excel. La solution optimale est $(x_1^*,\, x_2^*) = (1{,}5\,;\, 3{,}5)$ avec $z{=}5$

Cette solution ne convient pas à (1) puisque x_1^* et x_2^* ne sont pas entiers...Il faut donc aller en 6. Séparation : on crée deux noeuds fils ; le noeud fils gauche avec la contrainte $x_1 \leq 1$ et le noeud fils droit avec la contrainte $x_1 \geq 2$

Nous allons donc maintenant résoudre N_1, le noeud fils gauche :

$$\begin{aligned} \max \quad & x_1+x_2 \\ & -x_1 + x_2 \leq 2 \\ & 8x_1 + 2x_2 \leq 19 \\ & x_1, x_2 \geq 0 \\ & x_1 \leq 1 \end{aligned}$$

Problème linéaire que l'on résout facilement : $(x_1^1, x_2^1) = (1\,;\,3)$ et z=4. 4 devient donc la borne inférieure pour z^* et on stocke (x_1^1, x_2^1).

Comme x_1^1 et x_2^1 sont entiers (donc faisable pour (1)) l'algorithme termine dans cette partie de l'arbre (on dit que *l'arbre a été taillé par respect des contraintes en nombres entiers*).

On s'intéresse donc maintenant au noeud fils droit N_2 qui est :

$$\begin{aligned} \max \quad & x_1+x_2 \\ & -x_1 + x_2 \leq 2 \\ & 8x_1 + 2x_2 \leq 19 \\ & x_1, x_2 \geq 0 \\ & x_2 \geq 2 \end{aligned}$$

La solution est $(x_1^2, x_2^2) = (2\,;\,1{,}5)$ et z=3,5 ; on remarque que la valeur de la fonction objectif est inférieure à z_U=4 mis à jour en N_1, l'algorithme s'arrête donc. On dit que *l'arbre a été taillé par bornes.*

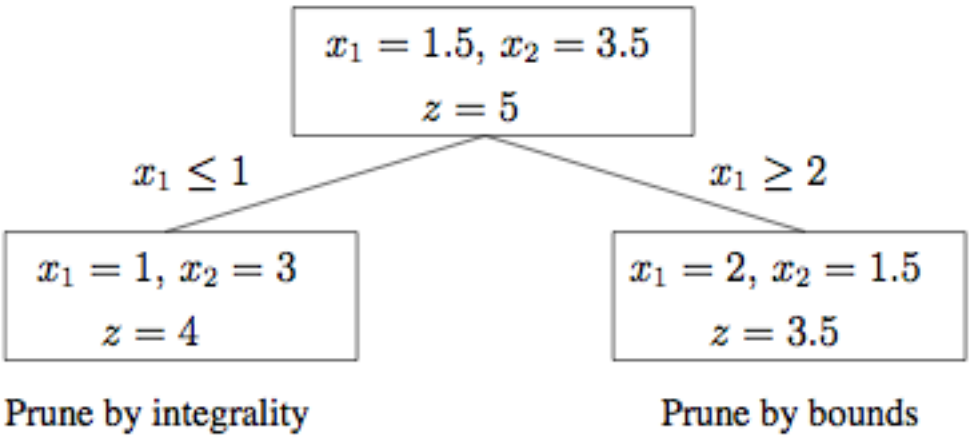

FIGURE 7.1 – Arbre Branch and Bound

7.3 Exemple 2

Le principe est exactement le même que l'exemple 1 mais permet de voir d'autres réactions de l'algorithme sur l'exemple suivant :

$$\begin{aligned} \max \quad & 3x_1+x_2 \\ & -x_1 + x_2 \leq 2 \\ & 8x_1 + 2x_2 \leq 19 \\ & x_1, x_2 \geq 0 \\ & x_1 \leq 1 \end{aligned}$$

Ici le noeud N_2 a une solution z supérieure à celle de N_1 donc on continue à séparer... sauf que pour le noeud fils droit N_4 les contraintes $x_1 \geq 2$et $x_2 \geq 2$ ne respectent pas $8x_1 + 2x_2 \leq 19$ et l'algorithme s'arrête ! On dit que *l'arbre a été taillé par infaisabilité.*

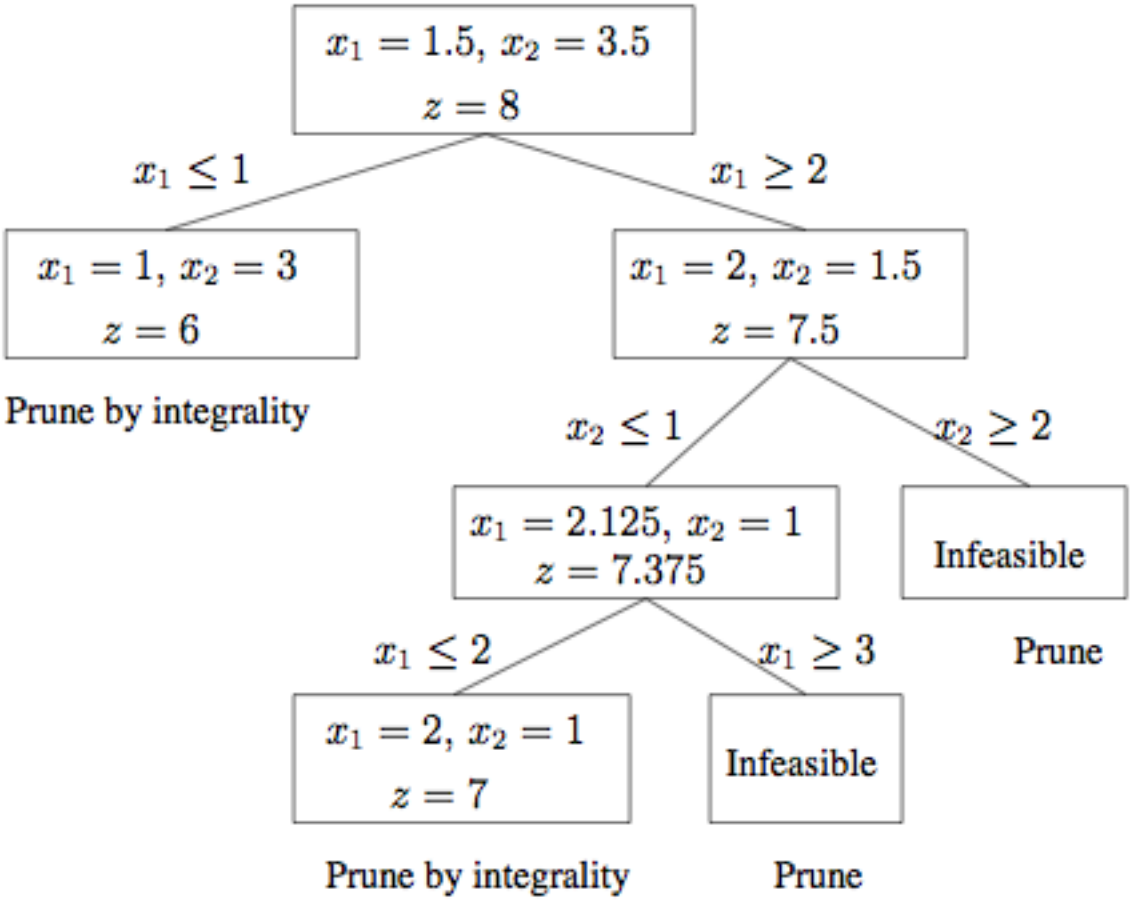

FIGURE 7.2 – Arbre Branch and Bound

7.4 Stratégie de séparation

7.4.1 Séparation forte

A chaque noeud N_i, il faut choisir sur quelle variable on va imposer les contraintes $x_i^k \leq \lfloor x_i^k * \rfloor$ ou $x_i^k \geq \lceil x_i^k * \rceil$. Dans l'exemple 1, pour N_1, x_1 et x_2 ne sont pas entiers et j'ai séparé aléatoirement sur la première variable. On comprend bien que sur des problèmes gigantesques, savoir sur quel variable de décision se fait la séparation est crucial pour réduire la taille de l'arbre de recherche (les problèmes MILP ont sur des problèmes pratiques actuelles des centaines de variables de décisions et autant de contraintes). Comme on a pu le voir à travers les exemples, le fait de descendre dans l'arbre de recherche va forcément détériorer la solution optimale (par ajout de contraintes à un problème linéaire). On se propose de rendre cette "détérioration" maximale de manière à avoir une borne inférieure la plus haute possible et ainsi "tailler" le plus de branche possible (cf point 5. de l'algorithme). Considérons le noeud N_i où pour j=1...p x_j^i (la variable de décision x_j dans le vecteur optimal à N_i noté x^i, vous suivez toujours ?). On définit donc N_{ij}^- le noeud N_i où on a ajouté la contrainte $x_j \leq \lfloor x_j^i * \rfloor$. On lui associe D_{ij}^- la détérioration (positive !) de la fonction objectif entre N_i et N_{ij}^-.

On définit de même N_{ij}^+ avec $x_j \geq \lceil x_j^i * \rceil$. On lui associe D_{ij}^+ la détérioration (positive !) de la fonction objectif entre N_i et N_{ij}^+.

On peut soit viser à maximiser $\min(D_{ij}^-, D_{ij}^+)$ - les deux valeurs sont donc prises en compte à coup sûr - ou $D_{ij}^- + D_{ij}^+$. Dans les algorithmes les plus récents et performants, considère une combinaison convexe de ces deux critères, avec un poids plus important sur le premier.

Si, comme je viens de l'exposer, on fait ce calcul pour tous les noeuds N_i et tous les j tels que x_j^i n'est pas entier, on comprend bien que cela représente énormément de calculs ; on appelle cette stratégie la *séparation forte "complète"* (ou *séparation forte* dans le cas d'un nombre restreint de noeuds sur lesquels on la pratique). Comme on le verra plus tard, cette stratégie permet de réduire d'environ 20 la taille de l'arbre de recherche par rapport à une séparation hasardeuse.

La résolution de N_{ij}^+ et N_{ij}^- se fait par la méthode du simplexe dual ; ce sont des faibles variations de N_i en termes de contraintes.

7.4.2 Most infeasible Branching

Il est également mentionné dans *Optimization Methods in Finance* qu'il peut être utilie de se concentrer (avec un certain seuil) sur la séparation forte sur un sous-groupe de variables de décisions : celles dont la partie non entière est la plus proche de 0,5. Les résultats présentés dans *Branching Rules Revisited* montrent que les performances d'un tel algorithme sont très proches d'une séparation aléatoire et n'a que peu voire pas d'intérêt.

7.4.3 Séparation par pseudocoûts

Un peu de vocabulaire Si la méthode de séparation forte permet de fortement réduire le nombre de noeuds de l'arbre, le temps de calcul est quant à lui significativement augmenté. On pense donc naturellement à *estimer* D_{ij}^+ et D_{ij}^-. On va donc s'intéresser aux pseudocoûts.

Définissons d'abord $f_i^j = x_i^j - \lfloor x_j^i \rfloor$; on ne s'intéresses qu'aux variables de décision qui ne sont pas encore entières, on peut donc poser

$P_i^- = \frac{D_{ij}^-}{f_i^j}$ et $P_i^+ = \frac{D_{ij}^+}{1-f_i^j}$

Ces pseudocoûts restent quasi constants dans tout l'arbre. Ils sont initialisés grâce à la séparation forte et sont estimés ensuite : pour P_i^- on prend la moyenne des $\frac{D_{ij}^-}{f_i^j}$ aux noeuds pour lesquels on a séparé sur x_j.

Au noeud N_i, on considère les estimations de P_i^- et P_i^+, on calcule f_j^i puis D_{ij}^- et D_{ij}^+ par la formule donnée plus haut. La variable sur laquelle on va séparer est celle qui maximisera une fonction de "score", comme $\min(D_{ij}^-, D_{ij}^+)$ ou $D_{ij}^- + D_{ij}^+$ ou les deux. Comme on le verra plus tard, cette méthode a le chic d'être en général plus rapide que la séparation forte mais l'arbre final comportera un nombre significatif de noeuds supplémentaires.

7.4.4 Hybride

Comme on l'a indiqué à la section précédente, il est courant d'initialiser la méthode des pseudocoûts avec une ou plusieurs itérations de la séparation forte. La séparation hybride s'intéresse à la profondeur d de l'arbre jusqu'à laquelle il est intéressant de pratiquer la séparation forte *puis* la méthode des pseudocoûts.

7.4.5 Reliability branching

Cette nouvelle méthode de séparation a été introduite en 2004 par un groupe de chercheurs du ZIB dans *Branching Rules Revisited* et se base sur la méthode des pseudocoûts :

La méthode des pseudocoûts garde une trace des noeuds auxquels x_i a été séparée et calcule la moyenne de l'augmentation de la fonction objectif lorsqu'on a séparé sur x_i. La nouvelle méthode prétend qu'en-dessous d'un certain seuil (le nombre de noeuds en mémoire), cette méthode n'est pas fiable (faire une moyenne sur deux chiffres...).

La première étape consiste à classer les variables de décision non entières par ordre décroissant sur j $\in$ C de $\min(D_{ij}^-, D_{ij}^+)$ au noeud N_i. On cherche ensuite les variables de décision qui ne vérifient pas le critère de fiabilité (i.e. dont la moyenne est calculée sur peu de valeurs) dans un ensemble F. Sur ces variables, on pratique *en partie* l'algorithme de séparation forte en limitant le nombre d'itérations pour le calcul du changement de valeur de la fonction objectif. C'est ce qu'on appellera le paramètre λ. Après ces λ itérations, on a, pour toutes les variables de décision F, une nouvelle valeur de D_{ij}^- et D_{ij}^+, on peut donc les replacer dans la liste des $\min(D_{ij}^-, D_{ij}^+)$ au noeud N_i. Il suffit enfin, qu'il ait changé ou pas, de séparer sur l'indice vérifiant : $max_{j \in C}$ $\min(D_{ij}^-, D_{ij}^+)$. Si le critère de fiabilité est vérifié, on utilise la méthode *classique* des pseudocoûts.

On pourra se rapporter à *Branching Rules Revisited* pour de plus amples détails. Cependant, j'ai cru bon de rapporter la preuve du fondement de cette méthode, qui devient optimale pour $\lambda = 4$ et $\eta = 8$ (critère de fiabilité).

Autres méthodes De nombreuses autres méthodes existent et se sont développées très récemment : propagation du domaine, branching on nonchimerical fractionalities, cloud branching ...

7.5 Sélection du noeud

On dispose d'un arbre de recherche. On se demande quel noeud N_i on va choisir pour continuer l'exploration (on se demande tout simplement quel méthode d'exploration de l'arbre binaire sera la plus rapide) ...Il faut donc définir ce critère de rapidité. En fait, il y en a deux ; soit on cherche à diminuer le plus rapidement possible la borne supérieure de notre solution : *best feasible* (et ainsi éliminer plus de noeuds dont la fonction objectif est très détériorée), soit on cherche à augmenter le plus rapidement possible la borne inférieure de la solution : *depth first* (et ainsi prouver que l'actuelle meilleure solution est optimale).

7.5.1 Best node

On utilise une estimation basée sur les pseudocoûts ; on pose avec les notations précédentes :

$E_i = z_i + \sum_{j=1}^{p} \min(D_{ij}^-, D_{ij}^+)$

Il ne reste plus qu'à choisir l'indice i minimisant E_i pour faire l'effet "inverse" de la séparation forte et diminuer la borne supérieure de la fonction objectif.

7.5.2 Depth first

Cette méthode va dans le même sens que la séparation forte : on cherche à augmenter la borne inférieure de la solution : de combien au minimum sera détériorée la fonction objectif à la fin de l'algorithme ? Comme son nom l'indique, elle va d'abord s'intéresser aux feuilles de l'arbre en y "plongeant" : on s'intéresse aux noeuds fils à chaque itération et pas aux noeuds voisins. Cette

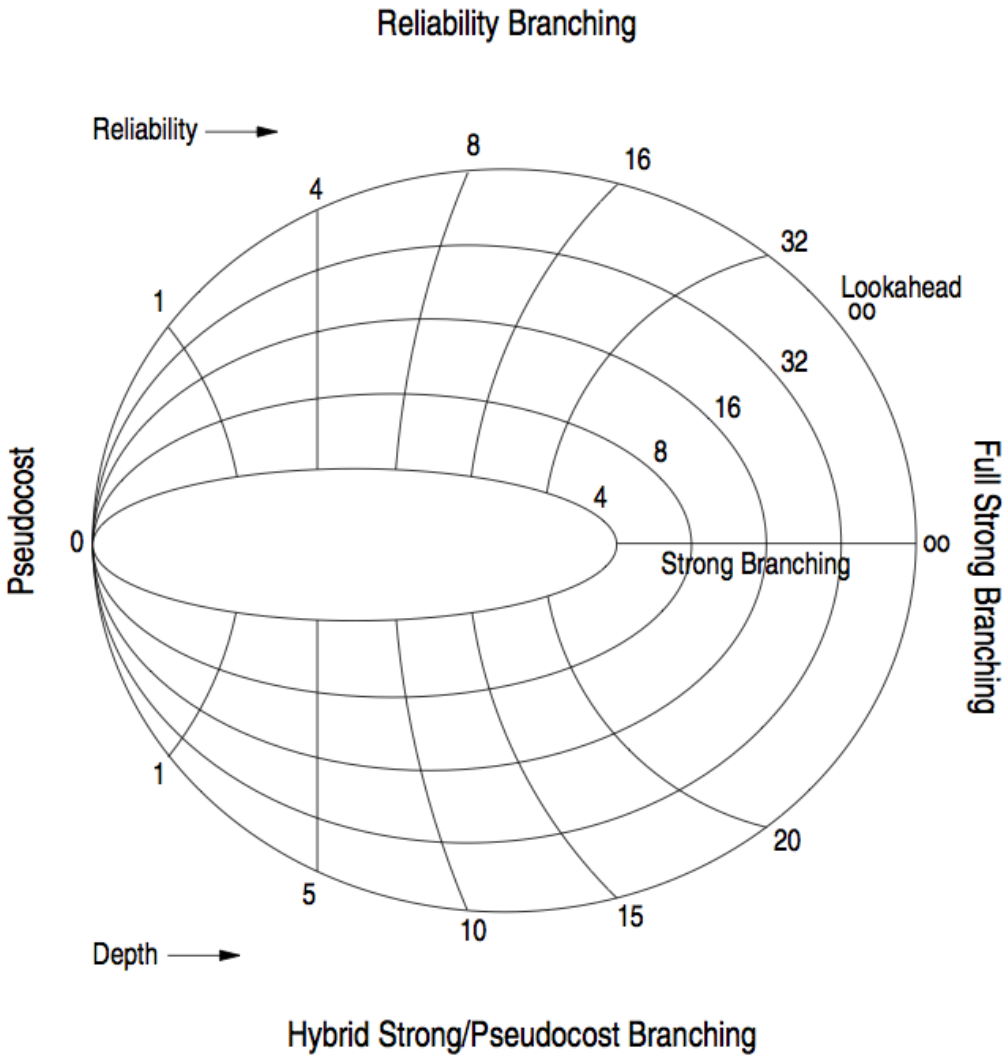

FIGURE 7.3 – Relations entre les méthodes hybrides, reliability et séparation forte

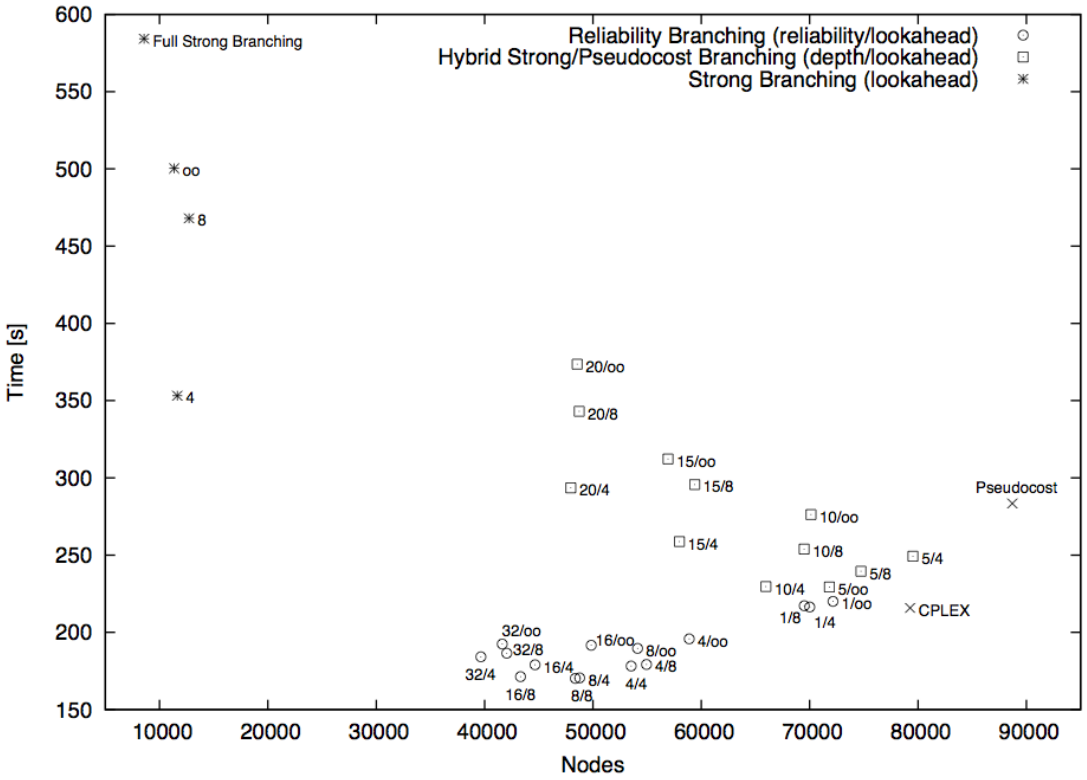

FIGURE 7.4 – Comparaison entre les méthodes hybrides, reliability et séparation forte : temps de calcul en fonction du nombre de noeuds de l'arbre de recherche final.

solution est extrêmement rapide car chaque nouvelle itération n'est qu'un léger changement dans les contraintes, donc en initialisant l'algorithme avec les résultats trouvés par le noeud père, la méthode du simplexe dual est très rapide. De la même manière que la séparation forte réduit fortement la taille de l'arbre et le nombre de noeuds, cette méthode de sélection des noeuds est environ 10 fois plus rapide qu'une méthode naïve. En revanche, elle peut faire des calculs inutiles : on peut se perdre dans des parties de l'arbre qui n'auraient pas été explorées car la fonction objectif est bien loin de sa borne inférieure.

7.5.3 Heuristics

Les heuristiques fournissent une estimation de la fonction objectif et se focalisent sur sa borne supérieure. On peut alors réduire le temps de calcul mais aussi fournir une réponse (peut-être fausse) si l'algorithme termine à cause d'une contrainte temporelle. Dans les exemples déjà donnés, l'heuristique est par exemple l'estimation E_i.

7.5.4 Feasibility Pump

Tandis que la méthode Cutting Planes que nous verrons dans le chapitre suivant s'efforce de trouver une solution optimale rapidement, la technique Feasibility Pump s'efforce quant à elle de trouver une solution admissible rapidement. Schématiquement, supposons que l'on possède un vecteur x_* qui vérifie les contraintes linéaires et un vecteur x^* qui vérifie les contraintes en nombre entier. Cette méthode "pompe" x_* "vers" x^* pour que la solution devienne entière.

En pratique, la méthode est la suivante :

Initialisation x_* solution de $\{x : Ax \leq \mathrm{b}\}$

Répéter Pumping cycle
Arrondir solution entière $x^* = \lfloor x_* \rfloor$

Retourner x_*

Chapitre 8

Cutting planes

Algorithme On s'intéresse toujours au problème (1) défini plus haut. On introduit des variables d'écart et on sépare les variables entières dans un sous-ensemble I des variables continues dans un sous-ensemble C. Les contraintes sont donc :

$$\begin{array}{cccc} z_I & = & \min & c^T x \\ & Ax & = & b \\ & x & \geq & 0 \\ & x_j \text{ entier} & \text{pour} & \text{j} \in I \ldots, \text{p} \end{array}$$

On a donc $\sum_{j \in I} a_j x_j + \sum_{j \in C} a_j x_j = \text{b}$
et on définit pour b et j $\in$ I f_0 et f_j les parts non entières de b et a_j : $a_j = \lfloor a_j \rfloor + f_j$
On a donc $0 \leq f_j < 1$ et en remplaçant on a donc :
$\sum_{j \in I et f_j \leq f_0} f_j x_j + \sum_{j \in I et f_j > f_0} (f_j\text{-}1) x_j + \sum_{j \in C} a_j x_j = \text{k} + f_0$
où k est entier.
En distinguant le cas k positif et le cas k négatif on obtient :
$\sum_{j \in I et f_j \leq f_0} \frac{f_j}{f_0} x_j$ - $\sum_{j \in I et f_j > f_0} \frac{(f_j - 1)}{f_0} x_j + \sum_{j \in C} \frac{a_j}{f_0} x_j \geq 1$
ou
- $\sum_{j \in I et f_j \leq f_0} \frac{f_j}{1 - f_0} x_j$ - $\sum_{j \in I et f_j > f_0} \frac{(f_j - 1)}{1 - f_0} x_j + \sum_{j \in C} \frac{a_j}{1 - f_0} x_j \geq 1$

On peut voir ces deux inégalités comme deux inégalités de produits scalaires avec $< a^1 \mid x > \geq 1$ et $< a^2 \mid x > \geq 1$ donc $\sum \max(a_j^1, a_j^2) x_j \geq 1$

Pour chaque terme des inégalités ci-dessus, il suffit donc de prendre le terme positif pour prendre le max, ce qui donne :

$\sum_{j \in I et f_j \leq f_0} \frac{f_j}{f_0} x_j + \sum_{j \in I et f_j > f_0} \frac{(1 - f_j)}{1 - f_0} x_j + \sum_{j \in C et a_j > 0} \frac{a_j}{f_0} x_j$ - $\sum_{j \in C et a_j < 0} \frac{a_j}{1 - f_0} x_j \geq 1$

Cette inégalité est vrai pour tout b non entier et tout x_j entier i.e. pour tout j $\in$ I.

On ajoute cette inégalité aux contraintes du problème et on réduit ainsi la taille de domaine de solutions comme on le voit dans l'exemple qui suit. La preuve de la convergence de cet algorithme est disponible dans *Localization and Cutting-plane Methods* et *Integer Programming.*

Exemple On considère :

$$\begin{array}{cc} \max & \text{z}=x_1+x_2 \\ & \text{-}x_1 + x_2 \leq 2 \\ & 8x_1 + 2x_2 \leq 19 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \text{ entiers} \end{array}$$

On ajoute alors des variables d'écart :

$$\begin{aligned} \max \quad & z = x_1 + x_2 \\ & -x_1 + x_2 + x_3 = 2 \\ & 8x_1 + 2x_2 + x_4 = 19 \\ & x_1, x_2, x_3, x_4 \geq 0 \\ & x_1, x_2, x_3, x_4 \text{ entiers} \end{aligned}$$

Par la méthode du simplex, le tableau final est le suivant :

$$\begin{aligned} z + 0{,}6x_3 + 0{,}2x_4 &= 5 \\ x_2 + 0{,}8x_3 + 0{,}1x_4 &= 3{,}5 \\ x_1 - 0{,}2x_3 + 0{,}1x_4 &= 1{,}5 \\ x_1, x_2, x_3, x_4 &\geq 0 \end{aligned}$$

On a $x_3 = x_4 = 0$ et $x_1 = 1{,}5$ et $x_2 = 3{,}5$

On s'intéresse maintenant à une contrainte du problème, par exemple

$x_2 + 0{,}8\, x_3 + 0{,}1x_4 = 3{,}5$

On calcule les f_i : $f_0 = 0{,}5$; $f_1 = f_2 = 0$; $f_3 = 0{,}8$; $f_4 = 0{,}1$ L'inégalité donnée plus haut ajoute la contrainte :

$\frac{1-0,8}{1-05}\, x_3 + \frac{0,1}{0,5}\, x_4 \geq 1$ soit $2x_3 + x_4 \geq 5$

Pour des raisons pratiques, on peut exprimer cette inégalité en fonction de x_1 et x_2 :

$3x_1 + 2x_2 \leq 9$

Ce qui se représente graphiquement par :

On se doute alors qu'une nouvelle résolution du problème linéaire nous donnera une solution entière, vue la forme de l'espace S ainsi que le gradient de la fonction objectif. C'est le cas, puisque la solution optimale est, comme on peut s'en rendre compte sur le graphe, $x_1 = 1$ et $x_2 = 3$ (axes inversés sur la figure précédente).

Branch and cut Ce troisième algorithme se base sur les deux précédents. Dans l'arbre construit par la méthode Branch and Bound, on est amené à certains noeuds à pratiquer Cutting Planes pour améliorer la borne inférieure. . .C'est cet algorithme qui est implanté dans les logiciels de résolution les plus connus. Intéressons-nous plus particulièrement donc à ce qu'ils font dans la pratique : Cutting Planes est appliqué sur les premiers noeuds de l'arbre (pour en réduire sa taille) puis Branch and Bound est appliqué (pour sa vitesse de résolution). On peut voir les résultats spectaculaires de cette méthode en termes de réduction de taille de l'arbre dans *Solving Mixed Integer Linear Programs Using Branch and Cut Algorithm* by Shon Albert :

En conclusion, comme mentionné dans *Mixed Integer Linear Programming*, "No known strategy is best for all problems !".

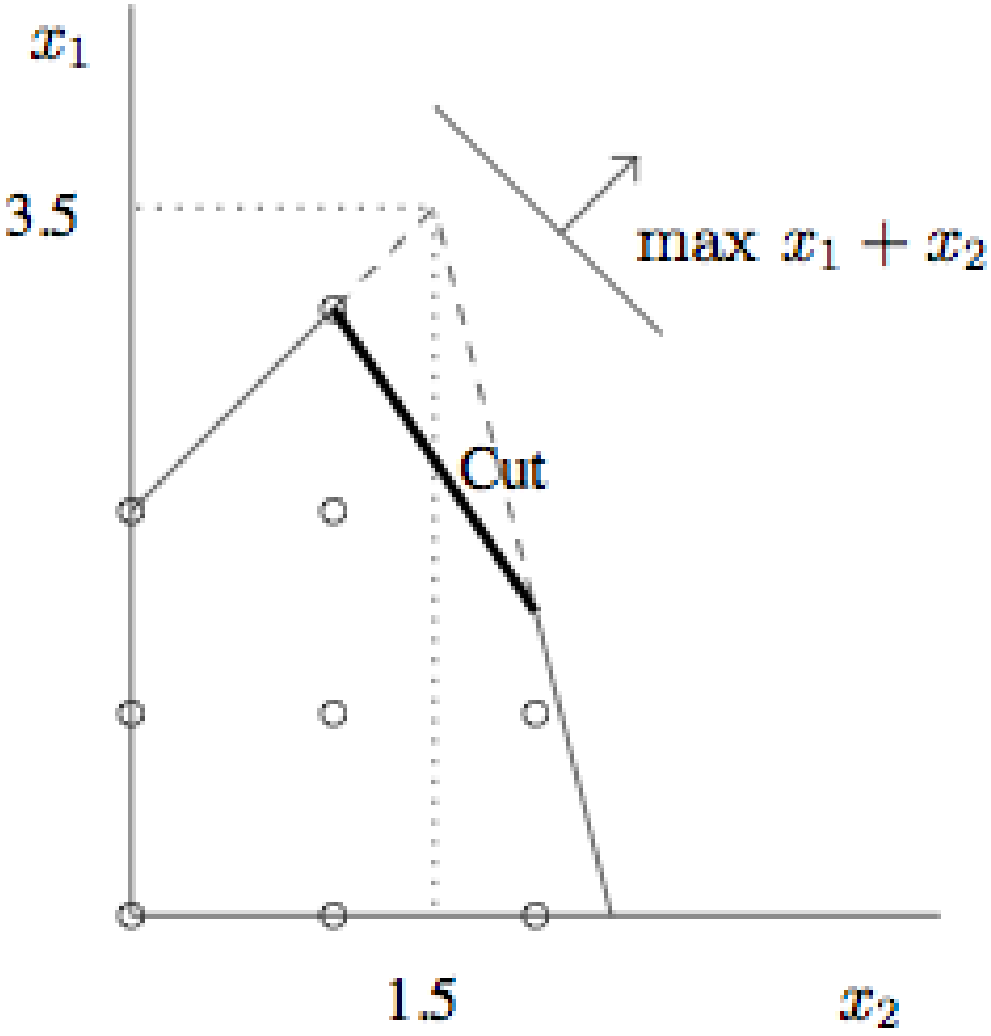

FIGURE 8.1 – Ajout de la contrainte de Gomory

Let n be odd.

$$\begin{aligned} \max \quad & z = -x_{n+1} \\ \text{s.t.:} \quad & 2x_1 + 2x_2 + \ldots + 2x_n + x_{n+1} \leq n \end{aligned}$$

Look at the following results when n=9.

Cuts	*Nodes*	*TotalNodes*
0	251	251
1	69	70
2	19	21
3	5	8

FIGURE 8.2 – Evolution de la taille de l'arbre avec le nombre d'itérations de Cutting Planes

Quatrième partie

Construction d'un index

Chapitre 9

Pourquoi ?

Cette partie est librement inspirée de *Optimization Methods in Finance* ainsi que du *Wall Street Journal* et notamment *A Portfolio That's as Simple as One, Two, Three Can This Index Fund Beat the Market ?*

Un débat récurrent en finance de marché est la gestion active ou passive d'un portfolio d'actions. Une gestion active se focalise sur des outils techniques, graphiques et analytiques, ainsi que sur la spéculation pour investir et constamment changer ses positions dans son portfolio. Au contraire, la gestion passive du portfolio consiste en une diversification importante des actifs dans lesquels on investit pour obtenir un retour sur investissement (une espérance) et un risque associé (une variance) précis. En finance, le retour sur investissement est toujours à relier au risque encouru lorsque l'on investit dans tel ou tel produit.

Il y a deux types de management passif. L'un consiste à choisir ses produits puis à les garder. L'autre, sur lequel nous allons nous focaliser, consiste à investir dans des indices. Ce qui nous amène à nous demander ce qu'est un indice, pourquoi y investir, et comment le construire. Premièrement, le but n'est pas ici d'analyser le marché pour trouver des produits mal pricés (trop chers ou peu chers par rapport au marché) mais à suivre l'évolution d'un marché spécifique (le marché des nouvelles technologies aux Etats-Unis pour le NASDAQ par exemple...). Ce marché se compose de grandes et de petites structures qui représentent un nombre gigantesque d'actions à suivre pour capter l'inertie du marché entier. Tout l'intérêt de la création d'un indice est de se limiter à une sous-population d'actions (de produits financiers plus généralement) qui reflétera le marché entier le plus fidèlement possible.
Comme on peut le voir actuellement, ces indices ont de plus en plus de succès. *Optimization Methods in Finance* l'explique pour plusieurs raisons :

- *Hypothèse d'efficience du marché financier :* on considère que l'information se répand instantanément sur le marché ; que toute information est connue de tous ; que les opérateurs réagissent correctement et instantanément aux informations qu'ils reçoivent. Autrement dit, à risque équivalent, tout produit financier doit avoir le même prix (il n'y pas de produits mal pricés), ce qui aurait rendu la stratégie "acheter et garder" plus efficiente que l'indice.
- *Performances constatées sur les marchés :* on a pu constater que les fonds d'investissement utilisant le management actif de portfolio avait souvent eu des succès moindres que les indices (en retour sur investissement). C'est d'autant plus vrai qu'il n'est pas rare que des fonds performants une année le soient beaucoup moins l'année d'après, contrairement à la régularité de l'investissement dans un indice.

— *Coût des transactions :* Le management actif de portfolio implique des coûts de transaction très importants dans la mesure où les positions doivent être modifiées très souvent. De plus, ce type de management consiste à trouver des produits intéressants (mal pricés ou dont les perspectives ou la conjoncture sont favorables) ; cela engendre d'énormes coûts en recherche (les banques d'investissement ont souvent un département entier consacré à cette recherche

Par ailleurs, un avantage important des indices mentionné dans *The Wall Street Journal - A Portfolio That's as Simple as One, Two, Three* est que l'on peut créer par soi-même un indice très simple avec seulement 3 produits ! Ce qui en fait un avantage certain pour les investisseurs particuliers qui ne scrutent pas la bourse tous les jours : il est par exemple conseillé d'investir à 40% dans les actions américaines, à 20% dans les actions internationales et à 40% dans tout type d'obligation. Ces trois produits, pour maximiser la diversification et minimiser le risque peuvent eux-même être pris parmi les indices reflétant ces marchés particuliers. Par exemple, concernant les actions américaines, il suffit d'investir 40% de la somme que l'on souhaite placer dans **iShares Core S&P Total U.S. Stock Market** ; concernant les obligations, on conseille par exemple de s'intéresser à **Vanguard Total Bond Market Index**. Bref, ce type de produits permet un investissement et une compréhension facile de l'investisseur. De plus, il est souvent nécessaire de corriger ses positions après un an (pour revenir au rapport actions/obligations initial), ce qui est bien plus simple lorsque l'on a que 3 produits !

On comprend dès lors l'intérêt porté à ces indices. On s'intéressera dans les parties suivantes de ce chapitre à ce qui nous permettra de construire ces indices i.e. comment choisir des actifs représentatifs d'une population plus grande de produits financiers. On verra que s'il est plutôt aisé de formuler le problème ainsi que l'algorithme permettant de se restreindre à une sous-population d'actifs, on atteindra rapidement deux limites :

— *Nombre de paramètres :* Comme dit précédemment, l'algorithme de résolution du problème d'optimisation résultant du problème de sélection des actions est *relativement* simple. En revanche, pour refléter un marché comme le marché américain, on doit s'intéresser à 50 000 produits, ce qui représente, comme on le verra, 250 000 contraintes entières... On sera donc amené à s'intéresser à différents algorithmes de résolution pour une convergence plus rapide.

— *Critères de sélection :* Une critique adressée récemment par un investisseur célèbre, *Joel Greenblatt*, fondateur du hedge fund New-Yorkais *Gotham Capital* et résumée dans *The Wall Street Journal - Can This Index Fund Beat the Market ?* est que aujourd'hui les poids des différents sous-jacents des indices dans lesquels on investit se font en fonction de la capitalisation boursière (nombre d'action multiplié par le prix de l'action) :

> "Market weighting tends to be an unconscious bet on growth," *Dave Nadig*, director of research at Index Universe

On s'intéressera donc naturellement au critère de sélection et on le fera varier dans la suite de cette partie.

Chapitre 10

Exercice simple : the lockbox problem

Nous allons voir ici un exemple simple de programmation binaire tel que se fera la construction d'un fonds indiciel. Une entreprise de VAD reçoit des chèques sur l'échelle des Etats-Unis, découpés en 4 régions. Du fait des services postal et bancaire, le client a rempli son obligation dès lors que le chèque et fait et envoyé alors qu'il se passe plusieurs jours avant que l'entreprise ne puisse utiliser l'argent. Il y a donc de l'intérêt perdu sur ce transit. L'entreprise se pose donc la question de la rentabilité de la création de 4 lockboxes, des bureaux destinés à la réception des chèques. On dispose du volume total échangé par région, du délai par région et par lockbox de réception, et donc de l'intérêt perdu :

From	L.A.	Cincinnati	Boston	Houston
West	60	120	180	180
Midwest	48	24	60	60
East	216	180	72	180
South	126	90	108	54

FIGURE 10.1 – Intérêts perdus

On se demande quels lockboxes doivent être ouverts pour minimiser les pertes (coûts de fonctionnement du lockbox + intérêts perdus) soit la fonction :

$60x_{11}+120x_{12}+180x_{13}+180x_{14}+48x_{21}+21x_{22}+60x_{23}+60x_{24}+216x_{31}+180x_{32}+72x_{33}+180x_{34}+126x_{41}+90x_{42}+10$

Concernant les contraintes, il est clair que chaque région des Etats-Unis devra être affectée à un seul lockbox d'où

$\sum_j x_{ij} = 1 \ \forall$ i

Il faut par ailleurs ajouter la contrainte qu'une région ne peut être assignée qu'à un lockbox ouvert :

$x_{1j}+x_{2j}+x_{3j}+x_{4j} \leq 4y_j \ \forall$ j (on a mis 4, on aurait pu mettre n'importe quoi de plus grand) Avec cette inégalité, si $y_j = 0$ aucune région ne peut être affectée au lockbox j.

La solution est obtenue à l'aide de Matlab :

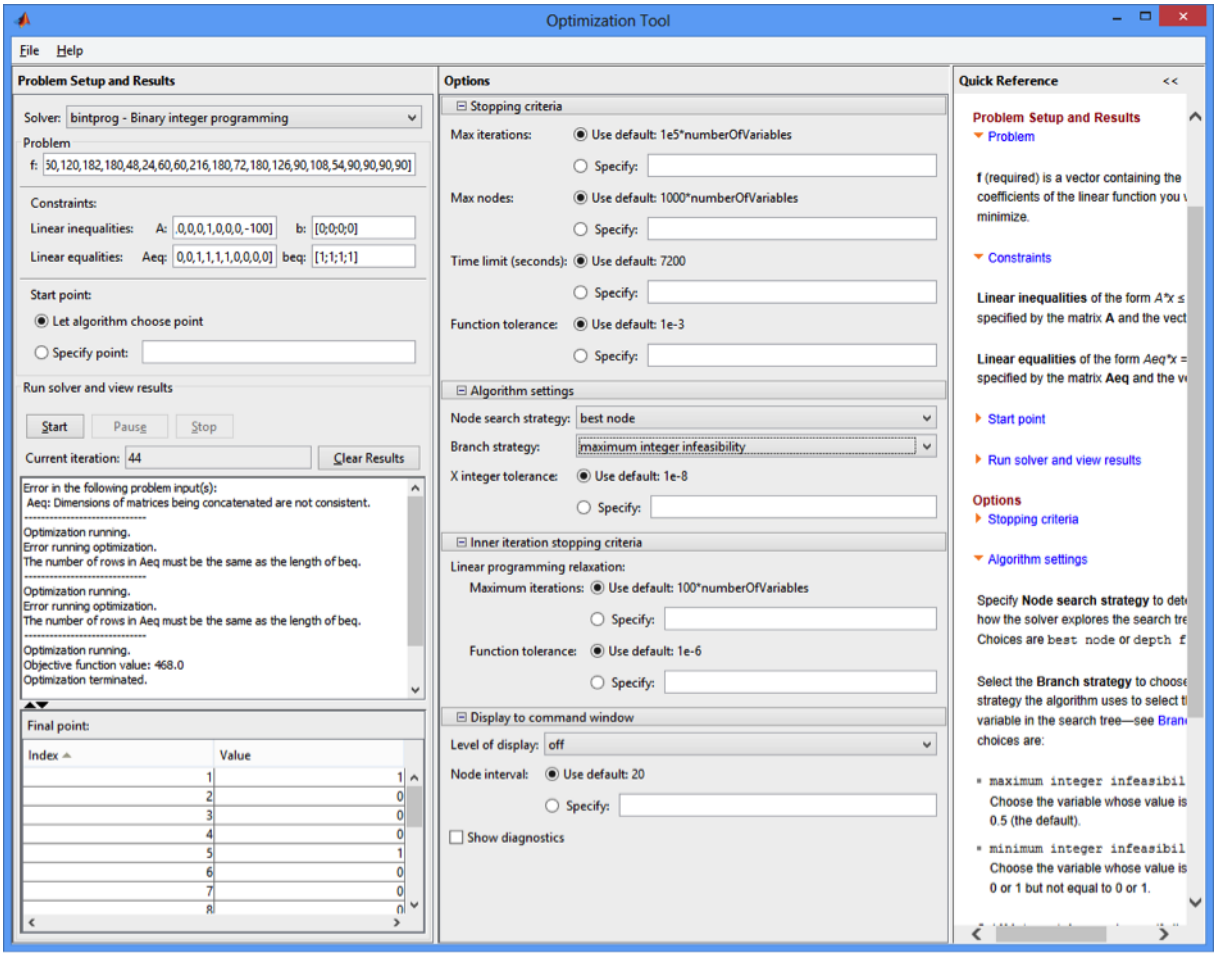

FIGURE 10.2 – Utilisation de Matlab pour la résolution des problèmes binaires

Chapitre 11

La suite

Ce premier compte-rendu est l'occasion de m'approprier le sujet de différentes manières. Premièrement, j'ai pu comprendre le contexte des fonds d'investissement en finance, ainsi que bien d'autres enjeux comme les Mutual Funds, le management actif et passif, le jeu des agences de notation et j'en passe comme je l'ai exposé en partie dans le chapitre précédent. Deuxièmement, j'ai pu acquérir les bases théoriques fondamentales en programmation linéaire en nombres entiers. La tâche fut plus théorique et difficile que la précédente, d'autant plus que beaucoup d'algorithmes sont très récents et la recherche continue dans ce sens. Par ailleurs j'ai aussi pu maîtriser LaTeXainsi que m'essayer à l'outil d'optimisation de Matlab et Excel ainsi que CPLEX, un solveur pour l'environnement AMPL, bien plus complet que ce que l'on peut trouver d'autre.

Dans la suite, je considèrerai ce compte-rendu "acquis" c'est-à-dire que l'on a maintenant compris quels sont les principaux algorithmes de résolution, quels sont ceux qui marchent le mieux et dans quel cas et pourquoi ils sont implémentés de cette manière dans les logiciels de résolution. Je n'ai pas à ma disposition le temps nécessaire pour réaliser un nouveau modèle. Je pourrais en revanche comparer les temps de calcul, les tailles des arbres de recherche, etc. selon les méthodes. Mais cet exercice est très fastidieux et nécessite un équipement spécifique.

Je vais donc me consacrer dans la suite du Projet Recherche, comme mentionné dans le chapitre précédent, aux strictes applications à la finance. Il s'agit donc de trouver un marché à refléter, trouver des informations sur ce marché (mesurer la corrélation ou toute autre mesure de "ressemblance" entre les actifs pour chaque produit financier de ce marché) et construire les contraintes, trouver un critère de pondération des actifs dans le portefeuille, résoudre le problème et suivre dans le temps les performances du portefeuille pour voir si oui ou non un index (donc managé passivement) peut surperformer le marché (et ce surtout avec très peu de risques et bien moins de frais que les fonds managés activement). Enfin, on s'intéressera à l'intérêt de revoir nos positions à différents intervalles de temps (mise à jour du critère de pondération) et les frais liés à ce repositionnement.

Le travail le plus intéressant dans cette partie est également celui sur lequel on trouve le moins de renseignement sur internet : la comparaison des méthodes de choix du critère de pondération. Comme mentionné plus haut, en général les actifs sont choisis par capitalisation (les plus grosses entreprises ont donc une place importante dans le fond) ou par prix par action mais de nombreux auteurs critiquent vivement ce choix (le potentiel de croissance, les dividendes, la rentabilité des entreprises doivent être pris en compte aussi - on pense notamment aux sociétés des NTIC qui font peu ou pas de bénéfices donc ne génèrent pas de dividendes, pourtant raison principale de la possession d'une action).

Bibliographie

[1] "Sas/or(r) 9.3 user's guide : Mathematical programming."

[2] A. Prior, "A portfolio that's as simple as one, two, three," *The Wall Street Journal*, 2013.

[3] M. Phillips, "Can this index fund beat the market ?" *The Wall Street Journal*, 2011.

[4] "The case for mutual funds."

[5] "Perfect your fund portfolio : When to sell funds."

[6] S. Boyd and L. Vandenberghe, "Localization and cutting-plane methods," 2003.

[7] S. O. Krumke, "Integer programming," 2006.

[8] G. GAMRATH, "Improving strong branching by propagation," 2012.

[9] P. Wood, "Non-capitalization weighted stock market index and index fund or funds," 2006.

[10] S. Albert, "Solving mixed integer linear programs using branch and cut algorithm."

[11] E. R.-C. Javier Larrosa, Albert Oliveras, "Mixed integer linear programming."

[12] N. E. SEK Ghen-Ly, CRESSOT Maxime, "Report : Optimization methods in finance project," March 2013.

[13] J. Borghoff, "Mixed integer programming : Algorithms and applications," May 2012.

[14] R. T. Gerard Cornuejolsn, "Optimization methods in finance," 2005.

[15] "Binary and mixed-integer programming."

[16] A. M. Tobias Achterberg, Thorsten Koch, "Branching rules revisited," May 2004.