République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Abderrahmane Mira, Béjaia
Faculté des Sciences Exactes
Département de Recherche Opérationnelle

Mémoire de Fin d'Études

Pour l'obtention du diplôme de Master

Option : Modélisation mathématique et techniques de décision

Thème :

Méthode hybride pour la résolution des problèmes de programmation linéaire en nombres entiers

Réalisé par :

Dihya AISSAOUI

Souad BENHAMA

Devant le jury composé de :

Encadreur	M^r M.O. BIBI	Professeur
Président	M^r B. BRAHMI	M.C.A
Examinatrice	M^{me} N.LAZARI	M.A.A
Examinateur	M^r N. KHIMOUM	M.A.A

Résumé :

Les problèmes d'optimisation linéaire en nombres entiers permettent de modéliser de nombreux problèmes réels difficiles à résoudre. Dans ce travail, les différentes méthodes exactes qui résolvent ces problèmes sont explicitement citées. En particulier, la méthode de coupes, la méthode de *Branch-and-Bound*, la programmation dynamique et la méthode de support à variables bornées. Les méthodes approchées sont aussi citées.

Ce mémoire s'intéresse à la mise en œuvre d'une méthode de résolution hybride pour obtenir des solutions de bonne qualité en un temps raisonnable. On combine la méthode de support à variables bornées avec une heuristique. Lors de processus de résolution, à chaque étape, un problème relaxé est résolu par la méthode de support et sa solution optimale est soumise à une procédure d'arrondissement judicieusement choisie. le concept le β-optimalité permet d'indiquer la qualité d'une solution approchée obtenue par cette heuristique.

Mots clés : Programmation linéaire en nombres entiers, méthodes exactes, solution approchée, méthode hybride, arrondissement.

Abstract :

Integer programming optimization problems allow us to model many real-life problems that are hard to solve. Such problems can be solved efficiently by using different methods that give us an exact solution. this type of methods are cited explicitly. Particularly, cutting plane approach, *Branch-and-Bound* approach, dynamic programming and support method with bounded variables. Approximative approches are also cited.

This work focuses on the implementation of an hybrid method to obtain a good solution in a reasonable time. We have combined support method with an appropriate heuristic. During the resolution process, at each step, the relaxated problem is solved by the support method and the obtained optimal solution is submitted to a rounding heuristic, judiciously chosen. The concept of β-optimality allows to indicate the quality of our approximate solution

Key words : Linear integer programming, exact methods, approximate solution, hybrid method, rounding.

Remerciements

Au terme de ce travail, nous tenons à exprimer dans ces lignes notre reconnaissance à toutes les personnes qui ont participé d'une manière ou d'une autre à son aboutissement.

En premier lieu, nous tenons à exprimer notre profonde gratitude à M^{r}. BIBI pour sa disponibilité, ses précieux conseils et le soutien qu'il nous a apporté pendant la réalisation de notre travail. Il n'a jamais manqué de nous conseiller et de nous orienter tout au long de notre mémoire. Qu'il trouvent ici l'expression de notre respect et de notre profonde reconnaissance.

Nous présentons nos respects et nos sincères remerciements aux membres du jury qui ont accepté d'évaluer notre travail.

Nous voudrions exprimer notre profonde reconnaissance et remerciements à nos parents qui ont tant sacrifié dans leurs vies, pour nous garantir une éducation et un avenir.

Nous ne pourrions clôturer ces quelques lignes sans témoigner nos plus vifs remerciements à nos ami(e)s pour leur aide et leur soutien moral.

Table des matières

Introduction générale **6**

1 Introduction à l'optimisation combinatoire **9**
1.1 Introduction . . . 9
1.2 Problèmes d'optimisation combinatoire . . . 10
1.2.1 Définition . . . 10
1.2.2 Complexité théorique d'un problème . . . 10
1.3 Algorithmes et outils de résolution . . . 13
1.3.1 Résolution exacte . . . 13
1.3.2 Résolution approchée . . . 14
1.4 Conclusion . . . 15

2 Programmation linéaire en nombres entiers **16**
2.1 Introduction . . . 16
2.2 Modélisation d'un problème de PLE . . . 17
2.3 Terminologie . . . 19
2.4 Formulation d'un problème de PLE . . . 20
2.5 Difficultés de la PLE . . . 22
2.6 Méthodes exactes pour la résolution des problèmes de PLE . . . 23
2.6.1 Méthode de coupes . . . 23
2.6.2 Méthode de séparation et d'évaluation . . . 29
2.6.3 Programmation dynamique . . . 36
2.6.4 Méthode directe de support pour la résolution d'un problème de PL à variables bornées . . . 38
2.6.5 Algorithme de résolution d'un problème de PLE par la méthode de support . . . 45
2.7 Conclusion . . . 54

3 Méthodes approchées pour la résolution des problèmes de PLE **55**
3.1 Introduction . . . 55
3.2 Un aperçu sur les heuristiques . . . 55
3.2.1 Pourquoi utiliser une heuristique ? . . . 56
3.2.2 Types basiques des heuristiques . . . 57
3.2.3 Comment évaluer une heuristique ? . . . 58
3.3 Heuristiques en PLE . . . 58
3.3.1 Heuristiques primales . . . 58
3.3.2 Heuristiques basées sur des relaxations . . . 59
3.3.3 Approches de recherche locale . . . 60
3.3.4 Métaheuristiques . . . 62

3.3.5 Heuristique basée sur la génération de colonnes 62
3.3.6 Algorithmes gloutons 64
3.4 Conclusion 67

4 Méthode hybride pour la résolution des problèmes de PLE 68
4.1 Introduction 68
4.2 Présentation de la méthode hybride 69
4.2.1 Heuristique d'arrondissement 70
4.2.2 Estimation de β-optimalité 72
4.3 Application de la méthode sur un exemple 74
4.4 Conclusion 81

Conclusion générale 82

Introduction générale

Depuis l'apparition de la révolution technologique, le monde a connu une croissance remarquable dans la taille et dans la complexité des organisations. Les petites boutiques d'artisanat à l'époque ont évolué pour devenir des sociétés d'un milliard de dollars d'aujourd'hui, ce qui a changé la segmentation des responsabilités de gestion dans ces organisations. Cependant, ce développement en expansion a engendré de nouveaux problèmes qui se posent encore à présent dans de nombreuses sociétés. Vu que les problèmes rencontrés doivent être résolus mais aussi pour rassurer la survie de l'entreprise d'une meilleure façon, cela a donné un environnement propice pour l'émergence de la recherche opérationnelle.

Une des caractéristiques de la recherche opérationnelle, est qu'elle tente souvent de rechercher la meilleure solution (appelée solution optimale) pour le modèle représentant le problème auquel on est confronté. Cette « recherche d'optimalité » est un thème important dans la RO. Ce processus est résumé en un terme dit "optimisation", si cette dernière est linéaire, il s'agit alors de la *programmation linéaire*. Celle-ci signifie que toutes les fonctions mathématiques de ce modèle doivent être des fonctions linéaires. Le mot programmation ici est synonyme de planification. Ainsi, la programmation linéaire implique la planification d'activités pour obtenir un résultat optimal, c'est-à-dire un résultat qui atteint le mieux le but spécifié parmi toutes les alternatives réalisables.

Le développement de la programmation linéaire est considéré parmi les plus importants progrès scientifiques. Son impact depuis 1950 a été extraordinaire. Aujourd'hui, c'est un outil standard qui a permis à de nombreuses entreprises d'économiser des milliers ou des millions de dollars dans les différents pays industrialisés du monde. Cependant, une limitation clé qui empêche l'application de cet outil est de trouver le plus souvent des valeurs non entières pour les variables de décision. Alors que dans de nombreux problèmes pratiques, les variables de décision n'ont de sens que si elles ont des valeurs entières. Par exemple, il est souvent nécessaire d'affecter des personnes, des machines et des véhicules à

des activités en quantités entières. Il s'agit alors d'un problème de programmation linéaire en nombres entiers [47].

Il apparait que les problèmes linéaires en nombres entiers soient relativement faciles à résoudre. Après tout, les problèmes de programmation linéaire peuvent être résolus de manière extrêmement efficace, la seule différence est que les problèmes linéaires en nombres entiers ont beaucoup moins de solutions à considérer. En réalité, l'ensemble des solutions réalisables est fini. En revanche, si on considère par exemple un cas simple de la programmation linéaire en nombres binaires avec n variables, il y a 2^n solutions à prendre en considération (alors que quelques solutions doivent être rejetées car elles sont pas admissibles). Ainsi, chaque fois que n est augmenté de 1, le nombre de solutions est doublé. Cela est dû à la complexité exponentielle de tels problèmes. Avec $n = 10$, il y a plus de 1000 solutions ; avec $n = 20$ il y a plus de 1,000,000 ; avec $n = 30$ il y a plus d'un milliard ; et ainsi de suite. Heureusement, les meilleurs algorithmes de la programmation linéaire entière d'aujourd'hui sont largement supérieurs à cette énumération exhaustive. Les problèmes qui nécessitaient un temps énorme pour leur résolution sont maintenant résolus en fraction de secondes en utilisant les meilleurs solveurs [47].

En général, les méthodes de résolution que l'on met en œuvre pour résoudre ce type de problèmes doivent prendre en considération deux facteurs : la qualité des solutions et le temps de résolution. Bien que ces deux facteurs soient généralement liés, parfois il est nécessaire de faire le choix entre trouver une solution (des solutions) optimale(s) ou de se contenter d'une solution approchée en un temps minime. Souvent, ce choix est influencé par la nature du problème traité. Afin de résoudre ces problèmes, on trouve trois classes principales : les méthodes exactes, les méthodes approchées (les heuristiques) et les méthodes hybrides. La définition d'une méthode de résolution exacte efficace, c'est-à-dire permettant d'obtenir une solution optimale du problème avec une complexité raisonnable, représente pour certaines classes de problèmes d'optimisation un objectif difficile à atteindre. Cela est en particulier le cas lorsque les instances sont de grande taille. La complexité de ce type de méthodes rend leurs applications généralement difficiles en pratique, voire impossibles, et cela malgré le développement incessant des outils informatiques. Devant ce constat, d'autres approches dites heuristiques (ou algorithmes approchés) sont apparues il y a plusieurs décennies. Ces algorithmes permettent dans la plupart des cas la génération d'une solution réalisable du problème rapidement mais le prix à payer est élevé.

Devant les limites rencontrées par les heuristiques pour obtenir une solution réalisable de bonne qualité pour certains types de problèmes, d'autres approches appelées métaheuristiques sont apparues. Ces algorithmes sont plus complets et complexes qu'une simple heuristique, et permettent généralement d'obtenir une solution de très bonne qualité. Le rapport entre la qualité de la solution finale et le temps d'exécution d'une métaheuristique reste cependant dans la majorité des cas très intéressant par rapport aux autres types d'approches de résolution [35] et [28].

Finalement une dernière classe d'algorithmes peut être trouvée. Où le principe consiste à combiner des algorithmes exacts et/ou des algorithmes approchés pour essayer de tirer profit des points forts de chaque approche et améliorer le comportement global de l'algorithme. La terminologie rencontrée dans la littérature est l'hybridation de méthodes.

L'objectif du travail présenté dans ce mémoire est de présenter une méthode hybride et de l'appliquer pour une instance d'un problème de programmation linéaire en nombres entiers. Cette approche efficace combine la méthode directe de support et une heuristique d'arrondissement pour obtenir des solutions de bonne qualité en un temps d'exécution « raisonnable ».

Ce travail s'articule autour de 4 chapitres : le premier est une introduction à l'optimisation combinatoire, au deuxième et au troisième , nous présenterons d'une manière unifiée la théorie et la méthodologie de la programmation linéaire en nombres entiers, nous citons les méthodes de résolution exactes ainsi que les heuristiques. Dans le dernier chapitre "Méthode hybride pour la résolution des problèmes linéaires en nombres entiers", nous allons présenter explicitement l'approche hybride à utiliser.

Chapitre 1

Introduction à l'optimisation combinatoire

1.1 Introduction

Les ingénieurs et les décideurs sont confrontés quotidiennement à des problèmes de complexité grandissante, qui surgissent dans des domaines très divers, comme dans la recherche opérationnelle, la conception des systèmes mécaniques, le traitement des images,etc. Le problème à résoudre peut souvent s'exprimer comme un *problème d'optimisation* : on définit une fonction objectif que l'on cherche à minimiser ou à maximiser par rapport à tous les paramètres concernés. La définition d'un problème d'optimisation est souvent complétée par la donnée de *contraintes* : toutes les composantes des solutions retenues doivent respecter ces contraintes, faute de quoi ces solutions ne sont pas réalisables.

En réalité, deux types de problèmes d'optimisation sont distingués : les problèmes discrets ou combinatoires et les problèmes continus. Dans notre mémoire, nous s'intéresserons à l'optimisation combinatoire. Des efforts ont longtemps été menés pour résoudre ce type de problèmes. L'optimisation combinatoire permet de résoudre un nombre important de problèmes grâce aux différentes méthodes qu'elle nous offre.

Les problèmes d'optimisation combinatoire sont le plus souvent faciles à modéliser, mais leur résolution est généralement difficile.

Dans ce premier chapitre, nous présentons d'abord quelques caractéristiques des problèmes d'optimisation combinatoire ainsi que leur complexité. Ensuite nous donnerons les algorithmes et outils de résolution.

1.2 Problèmes d'optimisation combinatoire

1.2.1 Définition

Généralement, on qualifie de « combinatoires » les problèmes dont la résolution se heurte à une explosion du nombre de combinaisons à explorer. C'est le cas par exemple lorsque l'on cherche à concevoir un emploi du temps : s'il y a peu de cours à planifier, le nombre de combinaisons à explorer est faible et le problème sera très rapidement résolu ; cependant, l'ajout de quelques cours seulement peut augmenter considérablement le nombre de combinaisons à explorer de sorte que le temps de résolution devient excessivement long.

Un problème d'optimisation est une maximisation (ou minimisation) d'une certaine fonction :

$$\max_{x \in S} f(x),$$

où :

- x : le vecteur constitué de variables de décision, avec $x = (x_1, x_2, ..., x_n)' \in \mathbb{R}^n$;
- f : la fonction objectif ;
- $x \in S$ ensemble des solutions réalisable,$\mid S \mid$ fini.

Si le problème consiste à minimiser f, alors il est équivalent de maximiser $(-f)$, car $\min f(x) = -\max(-f(x))$.

1.2.2 Complexité théorique d'un problème

Cette notion de problème combinatoire est formellement caractérisée par la théorie de la complexité qui propose une classification des problèmes en fonction de la complexité de leur résolution.

On entend ici par « complexité d'un problème » une estimation du nombre d'instructions à exécuter pour résoudre les instances de ce problème, cette estimation étant un ordre de grandeur par rapport à la taille de l'instance. Il s'agit là d'une estimation dans le pire des cas dans le sens où la complexité d'un problème est définie en considérant son instance la plus difficile [1]. Avant de passer aux classes des problèmes, il est nécessaire de se familiariser avec les termes suivants :

Définition 1.2.1 (Instance d'un problème d'optimisation combinatoire (OC))

Soit X un problème caractérisé par l'ensemble E de ses données, $E = (e_1, e_2, ..., e_n)$. Une instance de X serait alors caractérisée par un ensemble concret $E' = (e'_1, e'_2, ..., e'_n)$. Une deuxième instance serait caractérisée par un deuxième ensemble concret $E'' = (e''_1, e''_2, ..., e''_n)$, et ainsi de suite. L'invariant suivant est toujours vérifié : $|E| = |E'| = |E''|$.

Définition 1.2.2 (Problème de décision)

C'est un problème qui possède pour solution Oui/Non. On en déduit qu'un problème décisionnel abstrait fait correspondre à toute instance du problème l'ensemble des solutions vrai/faux.

Définition 1.2.3 (Problème d'optimisation)

Un problème d'optimisation est un problème caractérisé par un quadruplet $\langle I, S, f, mode\rangle$ où I représente les données du problème, S est le type de collection, ensemble ou le type d'éléments qu'on attend comme résultat, f est la fonction qui mesure la qualité du résultat et mode (max ou min) indique s'il s'agit de maximisation ou minimisation.

Définition 1.2.4 (Problème polynomial)

C'est un problème combinatoire pour lequel existe un algorithme polynomial pour le résoudre.

1.2.2.1 Classes de complexité

L'expérience montre que certains problèmes sont plus faciles à résoudre que d'autres, dans le sens où la meilleure solution peut être obtenue rapidement. La théorie de la complexité a été développée pour permettre à classer mathématiquement les problèmes selon leur difficulté [2]. On définit les trois grandes classes ***P***, ***NP*** et ***NP-Complet***. Le lecteur intéressé par de plus amples informations pourra consulter différents ouvrages dédiés à la complexité, dont les livres de Garey et Johnson [3] ou Papadimitriou [4] et plus récemment celui de Goldreich [5].

1.2.2.2 La classe *P*

Elle contient l'ensemble des problèmes polynomiaux, i.e., pouvant être résolus par un algorithme de complexité polynomiale. Cette classe caractérise l'ensemble des problèmes que l'on peut résoudre « efficacement ». On considère les problèmes appartenant à cette classe comme étant efficients ou traitables, ils peuvent être résolus par une *machine de Turing déterministe* pendant un temps de calcul polynomial.

1.2.2.3 La classe *NP*

C'est la classe des problèmes de décision pour lesquels il est possible de vérifier en temps polynomial qu'une solution donnée est réalisable ; c'est-à-dire, on peut construire un algorithme polynomial qui est capable de vérifier si pour une solution donnée, la réponse au problème de décision est OUI ou NON. Aussi, tout problème de décision qui peut être résolu par un algorithme polynomial, donc appartenant à la classe P, appartient également à la classe NP, d'où $P \subseteq NP$.

1.2.2.4 La classe *NP-complet*

Certains problèmes NP apparaissent plus difficiles à résoudre dans le sens où l'on ne trouve pas d'algorithme polynomial pour le résoudre avec une machine de turing déterministe. Les plus difficiles de NP définissent la classe des problèmes *NP-complet* : un problème de NP est *NP-complet* s'il est au moins aussi difficile à résoudre que n'importe quel autre problème de NP, i.e., si n'importe quel autre problème de NP peut être transformé en ce problème par une procédure polynomiale.

Cette classe contient un très grand nombre de problèmes. En effet, pour qu'un problème soit *NP-complet*, il faut qu'il soit dans la classe NP,i.e., que l'examen de chaque cas puisse être réalisé efficacement, par une procédure polynomiale. Si on enlève cette contrainte d'appartenance à la classe NP, on obtient la classe plus générale des problèmes *NP-difficiles*, contenant l'ensemble des problèmes qui sont "au moins difficiles" que n'importe quel problème de NP.

1.2.2.5 Démontrer la NP-complétude d'un problème

La démonstration d'appartenance d'un problème à la classe des problèmes *NP-complets* est très importante. En effet, une fois cette démonstration faite, on peut considérer comme peu vraisemblable l'existence d'un algorithme polynomial pour résoudre ce problème de manière exacte. Prouver qu'un problème de décision (P) est *NP-complet* comporte quatre étapes :

1. Montrer que *(P)* est dans *NP*.
2. Choisir *(P')* un problème *NP-complet* connu.
3. Établir une transformation f de $(P') \longrightarrow (P)$, polynomiale.
4. Prouver que f est bien une réduction polynomiale.

Ainsi, il est nécessaire de connaître des problèmes classiques, dont on sait qu'ils appartiennent à la classe NP-complet.

1.3 Algorithmes et outils de résolution

Nous avons vu précédemment qu'il existe des problèmes de différentes complexités. Les problèmes appartenant à la classe P ont des algorithmes polynomiaux permettant de les résoudre de manière exacte. Pour les problèmes appartenant à la classe NP-complet, l'existence d'algorithmes polynomiaux semble peu réaliste. Il existe pour certains problèmes NP-difficiles des méthodes de résolution exactes si on a de petites instances par exemple. Néanmoins, pour la plupart de ces problèmes, il est rare de pouvoir les résoudre avec une méthode exacte [2].

C'est pourquoi, des méthodes de résolution approchées, sont largement utilisées pour appréhender les problèmes NP-difficiles. Nous exposons dans cette partie les grandes lignes des méthodes les plus connues classées en deux catégories : les méthodes exactes et les méthodes approchées.

1.3.1 Résolution exacte

Le principe des méthodes exactes consiste à chercher la meilleure solution dans l'ensemble des solutions réalisables d'un problème.

L'optimisation exacte concerne toutes les méthodes permettant d'obtenir un résultat dont on sait qu'il est optimal à un problème précis. On peut les classer en trois grandes classes :

- La programmation dynamique ;
- La programmation linéaire et quadratique ;
- Méthode de séparation et évaluation.

1.3.2 Résolution approchée

Dans les méthodes exactes, le temps de calcul est généralement exponentiel, ce qui explique qu'elles ne sont utilisables que sur des problèmes de petite, voire de taille moyenne. Ainsi, pour les problèmes de grande taille, ces méthodes ne sont pas envisageables. Il est dans ce cas préférable d'utiliser des méthodes approximatives qui donnent des solutions approchées ou *sub-optimales*, c'est-à-dire qu'elles ne fournissent pas de solution optimale, mais essayent de s'en approcher. On choisit ainsi ces méthodes pour leur rapidité d'exécution. Les solutions obtenues peuvent ensuite servir de solutions initiales pour les méthodes amélioratrices.

Les algorithmes qui fournissent des solutions approchées sont appelés : les heuristiques et les métaheuristiques.

1.3.2.1 Les heuristiques

Une *heuristique* est un algorithme qui ne donne pas toujours de solution exacte. Elle est utilisée pour résoudre les problèmes de grande taille. Elle n'est pas fondée sur un modèle formel et elle n'aboutit pas nécessairement à une solution optimale.

De nombreuses heuristiques existent pour la résolution des problèmes d'optimisation. Certaines d'entre elles sont proposées pour la résolution d'un problème particulier lorsqu'elles prennent en compte des caractéristiques propres à celui-ci.

1.3.2.2 Les métaheuristiques

Les *métaheuristiques* sont des méthodes qui permettent de créer des *heuristiques*. Elles sont inspirées par des systèmes naturels. Elles sont d'inspirations diverses, comme le recuit

simulé [39], les algorithmes génétiques [11] ou les algorithmes de colonies de fourmis [40]. Elles sont en général de complexité exponentielle, tout en restant plus rapides que les méthodes exactes.

1.4 Conclusion

Dans ce chapitre, nous avons introduit l'optimisation combinatoire en donnant ses caractéristiques. L'optimisation des problèmes combinatoires est une optimisation discrète dans laquelle on cherche à trouver une solution dans un espace fini qui maximise (minimise) une fonction objectif. Nous avons introduit la théorie de la complexité, car la connaissance de la complexité d'un problème nous donne directement une idée sur l'outil à utiliser pour le résoudre. Pour la résolution des problèmes d'OC, nous avons présenté les deux classes majeures : les méthodes exactes et méthodes approchées. Les méthodes exactes ont l'avantage de garantir une solution optimale mais présente un inconvénient substantiel qui est un temps d'exécution important. En revanche, les méthodes approchées tentent de s'approcher de la solution optimale en un temps raisonnable.

Chapitre 2

Programmation linéaire en nombres entiers

2.1 Introduction

La programmation linéaire (PL) est un outil auquel on fait souvent appel dans de nombreux problèmes théoriques et pratiques. Dans le présent travail, nous présentons la programmation linéaire en nombres entiers qui sera abrégée en PLE dans la suite.

De nombreux problèmes peuvent être modélisés sous forme d'un problème d'optimisation dans lequel on cherche les valeurs des variables de décision pour lesquelles la fonction objectif prend une valeur maximale (ou minimale), ces variables étant soumises à un ensemble de contraintes. En pratique, on est constamment amené à traiter des problèmes de PL où certaines variables sont astreintes à prendre des valeurs entières ; on parle alors de programmation linéaire mixte. Si toutes les variables sont à valeurs entières, on a un problème de PLE. En particulier, les variables peuvent être simplement booléennes, c'est-à-dire ne prendre que les valeurs 0 ou 1. De nombreuses contraintes, en apparence non linéaires, peuvent être linéarisées grâce à des variables entières. Ces possibilités augmentent énormément le champ d'application de la programmation linéaire. Même si les programmes obtenus sont souvent difficiles à résoudre, la PLE est déjà très utile comme outil de modélisation [6].

Dans ce deuxième chapitre, nous commençons par modéliser un problème de PLE, tout en citant les spécificités de ce type de problèmes. Nous énoncerons la terminologie et toutes les définitions liées à de tels problèmes. Nous nous préoccuperons plutôt des techniques de résolution usuelles des problèmes de PLE. Nous nous limiterons aux méthodes de coupes, aux techniques de séparation et d'évaluation, la programmation dynamique et à la méthode de support.

2.2 Modélisation d'un problème de PLE

Commençons par quelques exemples :

Pour tout problème concret, il faut commencer par le mettre en équations, c'est-à-dire par effectuer la modélisation. Celle-ci comporte trois étapes :

1. Identification des variables ;
2. Formulation des contraintes ;
3. Définition de la fonction objectif.

Ensuite, Il faudra écrire le problème sous forme standard puis le résoudre par les "méthodes appropriées".

Exemple 2.2.1 (problème du sac-à-dos)

Le problème du sac-à-dos est un problème de sélection qui consiste à maximiser un critère de qualité sous une contrainte linéaire de capacité de ressource. Il doit son nom à l'analogie qui peut être faite avec le problème qui se pose au randonneur au moment de remplir son sac : il lui faut choisir les objets à emporter de façon à avoir un sac le plus utile possible, tout en respectant son volume, la question qui se pose est de savoir quels objets il doit mettre dedans.

On considère un ensemble d'objets étiquetés de 1 à n. Chaque objet $i \in \{1, ..., n\}$, dispose d'un poids w_i de valeur entière et d'un profit v_i. On dispose d'un sac dont le contenu ne peut excéder une capacité W entière. On désire le remplir de façon à maximiser le profit total des objets emportés, en respectant la contrainte de capacité.

Plus formellement, le problème noté (P), peut s'écrire sous forme d'un programme mathématique linéaire :

$$(P)\begin{cases} max\ z(x) = \sum_{i=1}^{n} v_i x_i, \\ s.c \quad \sum_{i=1}^{n} w_i x_i \leq W, \\ \qquad x_i \in \{0,1\}, i \in \{1, ..., n\}, \end{cases}$$

où le vecteur $x = (x_i, 1 \leq i \leq n) \in \{0,1\}^n$ est une solution du problème avec $x_i = 1$, si l'objet i est emporté dans le sac et $x_i = 0$ sinon.

On constate que la solution d'un tel problème ne peut prendre que deux valeurs 0 où 1, qui sont entières. Autrement dit, les objets sont indivisibles, et de ce fait on ne peut pas prendre une portion d'un objet dans le sac. Alors on peut caractériser le problème du sac-à-dos comme un problème de PLE.

Illustration

Soit un sac-à-dos et une liste d'articles de poids et de valeurs variées :

$$(P)\begin{cases} max\ z = 7x_1 + 3x_2 + 4x_3 + 3x_4, \\ s.c \quad 13x_1 + 8x_2 + 12x_3 + 10x_4 \leq 9, \\ \qquad x_i \in \{0,1\}, i = 1, ..., 4. \end{cases}$$

Après avoir résolu ce problème avec la méthode de programmation dynamique, nous obtenons comme solution : $x^* = (0, 1, 0, 0)$ avec, $z^* = 3$.

Exemple 2.2.2 (Fabrication des armoires et des tables)

Une entreprise fabrique des armoires et des tables telles que :

- Une armoire nécessite $1h$ de travail et $9m^2$ de bois ;
- Une table nécessite $1h$ de travail et $5m^2$ de bois ;
- On dispose de $6h$ de travail et de $45m^2$ de bois ;
- Chaque armoire génère un profit de $8u$, et chaque table $5u$;

La question qui se pose est quel est le nombre d'armoires et de tables qu'il faut fabriquer pour maximiser le profit de cette entreprise, i.e., $\max z = 8x_1 + 5x_2$, tout en respectant la contrainte de temps et la contrainte de la quantité du bois à utiliser : $x_1 + x_2 \leqslant 6$ et $9x_1 + 5x_2 \leqslant 45$. Il est bien clair que les deux variables x_1 et x_2 sont entières. le programme associé à ce problème est :

$$\begin{cases} max\ z(x) = 8x_1 + 5x_2, \\ s.c \quad x_1 + x_2 \leq 6, \\ \qquad 9x_1 + 5x_2 \leq 45, \\ \qquad x_1, x_2 \in \mathbb{N}. \end{cases}$$

Une grande variété de problèmes de la vie réelle dans la logistique, l'économie, les sciences sociales,etc, peuvent être formulés comme des problèmes d'optimisation linéaire en nombres entiers, comme le problème de budgétisation du capital, le problème de localisation des entrepôts, le problème du voyageur de commerce, le problème de sélection des machines, les problèmes des réseaux et des graphes tels que le problème de flux maximum et le problème de recouvrement,etc ([7]-[10]). Et aussi de nombreux problèmes d'ordonnancement peuvent également être résolus en tant que problèmes d'optimisation de PLE.

D'autre part, la solution d'un PLE relaxé (en relaxant la contrainte d'intégrité) ne donne pas directement une solution entière mais elle implique généralement des fractions. Par conséquent, nous pouvons trouver que la solution n'est pas réalisable, non pas à cause de l'échec à satisfaire les contraintes linéaires, mais à cause de ses valeurs fractionnaires. Dans ce cas, une question se pose :

- En supposant que la solution du problème relaxé ne soit pas un nombre entier, que peut-on faire à ce sujet ?

La réponse à cette question sera donnée dans la suite de ce travail.

2.3 Terminologie

Soit le problème de PL suivant sous forme standard :

$$(PL)\begin{cases} max\ z(x) = c'x, \\ s.c\ \ Ax = b, \\ \qquad x \geq 0. \end{cases}$$

1. **Programmation :** le terme de programmation faisant référence à l'idée d'organisation et de planification liée à la nature des phénomènes modélisés [12].
2. **« Résoudre »le problème (PL) :** trouver une solution optimale.
3. **Fonction économique :** le critère de qualité qui doit être maximisé (ou minimisé).
4. **Fonction linéaire :** somme de termes : *coefficient numérique* $\times$ *variable*. En particulier :
   - Elle n'a pas de terme constant ;

- Les seules opérations sur les variables sont : multiplication par un coefficient numérique et addition-soustraction.

5. **Variable :** une quantité
   - Dont la valeur n'est pas fixée à l'avance, mais qu'on peut choisir (variable de décision) ;
   - Qu'on convient de désigner par un nom (qui peut être quelconque).
6. **Variables d'écarts :** variables supplémentaires qui permettent de transformer des contraintes d'inégalités en contraintes d'égalités.
7. **Paramètre :** une quantité dont la valeur n'est pas fixée à l'avance, mais qui ne fait pas l'objet d'un choix.
8. **Contraintes :** prennent la forme d'équations et d'inéquations linéaires ainsi que les contraintes de non-négativité ($x_j \geq 0$).
9. **Solution(une décision) :** choix d'une valeur pour toutes les variables de décision.
10. **Solution réalisable :** si la solution respecte toutes les contraintes. L'ensemble des solutions réalisables est appelé "domaine réalisable ou admissible".
11. **Solution optimale :** si elle est à la fois réalisable et donne la meilleure valeur possible à l'objectif parmi toutes les solutions réalisables.
12. **Enveloppe convexe :** l'enveloppe convexe d'un sous-ensemble S de $\mathbb{R}^n$ est l'intersection de tous les sous-ensembles convexes de $\mathbb{R}^n$ contenant S. L'enveloppe convexe de S est par conséquent le plus petit sous-ensemble convexe de $\mathbb{R}^n$ qui contient S, on le note $conv(S)$.
13. **Algorithme :** une méthode de calcul qui permet par applications successives de la même opération (ou groupe d'opérations) d'aboutir au résultat.

2.4 Formulation d'un problème de PLE

Un problème de programmation linéaire (PL) est un problème d'optimisation consistant à maximiser (ou minimiser) une fonction linéaire dite fonction objectif sous contraintes linéaires exprimées sous forme d'équations ou d'inéquations. Un problème de PLE est un problème de PL dans lequel les variables sont astreintes à prendre des valeurs entières.

Un tel programme peut être représenté sous forme matricielle comme suit :

$$(PLE)\begin{cases}\max\ z(x) = c'x, \\ s.c \quad Ax \leq b, \\ \qquad x \in \mathbb{N}^n,\end{cases}$$

avec les notations suivantes :

- max : maximiser ;
- n : nombre de variables ;
- m : nombre de contraintes ;
- $A = (a_{ij},\ 1 \leq i \leq m, 1 \leq j \leq n)$: matrice réelle des contraintes, d'ordre $n \times m$;
- $c = (c_1, ..., c_n)'$: vecteur des coûts (profits), $c \in \mathbb{Z}^n$;
- $b = (b_1, ..., b_m)'$: vecteur colonne des seconds membres, $b \in \mathbb{Z}^m$;
- $x = (x_1, ..., x_n)'$: vecteur de variables de décision.

Dans le cas où les variables entières sont astreintes à ne prendre que les valeurs 0 ou 1, on parle alors de programmation linéaire en 0-1 ou en variables bivalentes, d'où le problème (PLB) suivant :

$$(PLB)\begin{cases}max\ z(x) = c'x, \\ s.c \quad Ax \leq b, \\ \qquad x \in \{0,1\}^{\mathbf{n}}.\end{cases}$$

Tout programme linéaire peut s'écrire sous sa forme standard, en ajoutant pour chaque contrainte une variable s_i appelée "variable d'écart", le programme linéaire s'écrivant alors de la manière suivante :

$$(PLS)\begin{cases}max\ z(x) = c'x, \\ s.c \quad (A+I)\binom{x}{x_s} = b, \\ \qquad x \in \mathbb{N}^n, x_s \in \mathbb{R}^m,\end{cases}$$

où I est la matrice identité d'ordre m et s est le vecteur des variables d'écart.

Avant de passer à la résolution d'un problème de PLE, on doit l'écrire sous forme

d'un programme linéaire continu (obtenu à partir de (PLE) en relaxant les contraintes d'intégrité) :

$$(PLR)\begin{cases} max\ z(x) = c'x, \\ s.c \quad Ax \leq b, \\ \qquad x_j \geq 0, \quad \forall j = 1, ...n. \end{cases}$$

2.5 Difficultés de la PLE

Parmi les difficultés des problèmes de PLE, on peut citer :

- L'optimum entier n'est pas forcément un sommet du polyèdre, il peut être à l'intérieure du polyèdre ;
- L'arrondi de la solution du problème relaxé n'est pas nécessairement optimal pour le problème de PLE, il pourrait même ne pas être réalisable ;
- Le polyèdre du problème relaxé peut être non vide, mais n'admettre aucune solution entière ;
- On peut construire des cas où les optima du problème de PLE et du problème relaxé sont aussi éloignés que l'on veut.

Les méthodes pour un problème de PL continu ne marchent donc pas pour un problème de PLE, puisqu'elles cherchent un sommet optimal du polyèdre qui, souvent, n'a pas de coordonnées entières.

En général, un problème de PLE est un problème difficile, on ne lui connaît pas d'algorithme polynomial. Même en ignorant la fonction objectif, trouver une solution entière à un système d'équations $Ax = b$ reste un problème difficile.

Actuellement, on peut résoudre des problèmes de PL à 100000 variables ; les chances de résoudre un grand problème de PLE dépendent beaucoup de la structure du problème. Typiquement, on parvient à traiter des programmes en 0-1 à quelques centaines de variables et des programmes comportant quelques dizaines de variables entières. Bien que des problèmes de PLE de taille supérieure soient facilement résolus dans certaines applications, il est clair qu'il faut prendre des précautions avant de lancer la résolution d'un grand problème de PLE avec un logiciel, par exemple faire des tests avec des exemples de taille croissante.

2.6 Méthodes exactes pour la résolution des problèmes de PLE

Un programme linéaire a une interprétation géométrique simple. Les éléments de l'ensemble $S = \{x \in \mathbb{N} \mid Ax \leq b\}$ sont les solutions réalisables du problème, et parmi ceux-ci, les éléments x^* qui maximisent le critère $c'x^*$ sont les solutions optimales. $Conv(S)$ est l'enveloppe convexe de l'ensemble S.

Il existe plusieurs méthodes exactes (algorithmes) de résolution des problèmes de programmation linéaire, qui permettent l'obtention d'au moins une solution optimale du problème à résoudre. Une fois une solution réalisable est obtenue, l'algorithme doit également être capable d'en prouver l'optimalité. Ce qui peut parfois être tout aussi difficile à faire que d'obtenir cette solution. Citons quelques approches exactes classiques utilisées en recherche opérationnelle que nous définirons juste après :

- Méthode de coupes ;
- Méthode de *Branch-and-Bound* ;
- Méthode de programmation dynamique ;
- Méthode de support.

2.6.1 Méthode de coupes

Le principe des méthodes de coupes est le suivant :
Nous commençons par résoudre (PLR). Si la solution optimale obtenue est un point à coordonnées entières, c'est terminé. Elle est la solution de notre problème initial à variables entières et le problème est résolu. Sinon (et c'est évidemment la situation la plus fréquemment rencontrée), on peut toujours tronquer le domaine des solutions en rajoutant une contrainte supplémentaire au problème de façon à éliminer ce point optimal sans exclure aucune solution entière. Une telle contrainte est appelée "*une coupe*" ou encore une : "*inégalité valide*".

Beaucoup d'inégalités valides peuvent donc être générées pour éliminer une solution fractionnaire donnée.

Après avoir rajouté une coupe (ou éventuellement plusieurs), le programme linéaire augmenté des contraintes est à nouveau résolu en continu par la méthode du simplexe. Comme l'ancienne solution reste duale réalisable, il est avantageux d'utiliser pour cela

l'algorithme dual du simplexe.

Si la solution optimale de ce nouveau problème est entière, c'est terminé : on a obtenu une solution optimale du problème en nombres entiers. Sinon, le raisonnement précédent peut être répété : on recherche une nouvelle coupe (éventuellement plusieurs) que l'on rajoutera à l'ensemble des contraintes ; puis le programme linéaire ainsi augmenté sera maximisé à nouveau, etc.

Si les coupes sont correctement choisies à chaque étape, le polyèdre initial se réduit jusqu'à coïncider avec l'enveloppe convexe des solutions entières, au moins au voisinage de la solution optimale. La solution continue du problème augmenté deviendra alors entière et le problème sera résolu.

2.6.1.1 Inégalités valides pour un programme linéaire

Définition 2.6.1 *[13] Une inégalité $\pi' x \leq \pi_0$ est une inégalité valide pour $X \subseteq \mathbb{R}^n$ si* $\pi' x \leq \pi_0 \quad \forall x \in X$.

Proposition 2.6.1 *[13] Si $\pi' x \leq \pi_0 \quad \forall x \in X$ est une inégalité valide pour $X = \{x : x \geq 0, Ax \leq b\} \neq \emptyset$, alors $\exists\ u \geq 0$ tels que $u'A \geq \pi'$ et $u'b \leq \pi_0$.*

2.6.1.2 Inégalités valides pour un problème de PLE

Inégalités valides de Chvàtal-Gomory [14]

Soient $u \in \mathbb{R}^m_+$, $S = X \cap \mathbb{Z}^n$, où $X = \{x \in \mathbb{R}^n_+ : \ Ax \leq b\}$, $A = (a_1, a_2, ..., a_n)$ une matrice d'ordre $m \times n$:

(i) L'inégalité suivante est valide pour X :

$$\sum_{j=1}^{n} u' a_j x_j \leq u' b,$$

car $u \geq 0$ et $\sum_{j=1}^{n} a_j x_j \leq b$.

(ii) L'inégalité suivante est valide pour X :

$$\sum_{j=1}^{n} [u' a_j] x_j \leq u' b,$$

car $x \geq 0$.

(iii) L'inégalité suivante est valide pour S :

$$\sum_{j=1}^{n}[u'a_j]x_j \leq [u'b],$$

car $x \in \mathbb{N}^n$, donc $\sum_{j=1}^{n}[u'a_j]x_j$ est entier.

Un résultat intéressant est que toute inégalité valide d'un problème de PLE peut être obtenue de cette façon.

Théorème 2.6.1 *[14] Si $\pi'x \leq \pi_0$ est une inégalité valide pour S avec π et π_0 entiers, alors elle s'obtient en un nombre fini d'itérations par la procédure de Chvàtal-Gomory.*

Ce résultat est soumis au bon choix de l'inégalité valide à chaque itération.

Exemple 2.6.1

Soit $S = X \cap \mathbb{Z}^n$ l'ensemble des points entiers dans X, où X est donné par :

$$\begin{cases} 5x_1 + 3x_2 \leq 8, \\ 2x_1 + 4x_2 \leq 6, \\ 7x_1 + x_2 \leq 15, \\ x_i \geq 0, \quad i = 1, 2. \end{cases}$$

(i) En combinant les contraintes avec les poids non négatifs $u = (\frac{3}{5}, \frac{1}{2}, \frac{2}{3})'$, on obtient l'inégalité valide pour X :

$$\frac{26}{3}x_1 + \frac{67}{15}x_2 \leq \frac{189}{5}.$$

(ii) En réduisant les coefficients du côté gauche de l'inégalité à l'entier le plus proche donne l'inégalité valide pour X :

$$8x_1 + 4x_2 \leq \frac{189}{5}.$$

(iii) Le côté gauche de l'inégalité est toujours entier pour tout point de S, on peut le réduire à l'entier le plus proche et on obtient ainsi l'inégalité valide pour S :

$$8x_1 + 4x_2 \leq 37.$$

2.6.1.3 Coupes de Gomory

L'une des coupes les plus étudiées et utilisées est la coupe de Gomory. C'est une coupe adaptée à la méthode du simplexe (et dual du simplexe). La détermination de cette coupe peut être expliquée simplement de la façon suivante :

Soit un problème de PLE sous forme standard :

$$(PLE)\begin{cases} max\ z(x) = c'x, \\ s.c \quad Ax = b, \\ \qquad x \geq 0,\ x_j \text{ entier}, \quad \forall j = 1, ...n. \end{cases}$$

Considérons le programme linéaire relaxé de (PLE) :

$$(PLR)\begin{cases} max\ z(x) = c'x, \\ s.c \quad Ax = b, \\ \qquad x_j \geq 0, \quad \forall j = 1, ...n. \end{cases}$$

Posons, pour une base J_B (avec $\mid J_B \mid = m$) et une solution réalisable x :

$$Ax = A_B x_B + A_N x_N = b,$$

où $x_B = (x_j, j \in J_B)$ et $x_N = (x_j, j \in J_N)$, $J = J_B \cup J_N = \{1, 2, ..., n\}$

Puisque A_B est régulière,

$$x_B + A_B^{-1} A_N x_N = A_B^{-1} b.$$

Posons :

$\bar{A} = A_B^{-1} A_N$ et $\bar{b} = A_B^{-1} b$.

On a donc :

$$x_B + \bar{A} x_N = \bar{b},$$

où $x_i + \sum\limits_{j \in J_N} \bar{a}_{ij} x_j = \bar{b}_i \quad (i \in J_B)$, $\bar{A} = (\bar{a}_{ij}, i \in J_B, j \in J_N)$ et $\bar{a}_{ij} = [\bar{a}_{ij}] + \{\bar{a}_{ij}\}$, tels que :

$[\bar{a}_{ij}]$: le plus grand entier inférieur à $\bar{a}_{ij}$;

$\{\bar{a}_{ij}\}$: la partie fractionnaire de $\bar{a}_{ij}$ telle que $0 \leq \{\bar{a}_{ij}\} < 1$.

Il s'ensuit :

$$x_i + \sum_{j \in J_N} ([\bar{a}_{ij}] + \{\bar{a}_{ij}\})x_j = [\bar{b}_i] + \{\bar{b}_i\}, \quad i \in J_B,$$

$$\Leftrightarrow x_i + \sum_{j \in J_N} [\bar{a}_{ij}]x_j - [\bar{b}_i] = \{\bar{b}_i\} - \sum_{j \in J_N} \{\bar{a}_{ij}\}x_j, \quad i \in J_B. \tag{2.1}$$

Soit $\tilde{x}$ une solution réalisable de (PLR) à valeurs entières. L'égalité (2.1) vaut pour $\tilde{x}$, le côté gauche étant à valeurs entières. De plus,

$$\tilde{x}_i + \sum_{j \in J_N} [\bar{a}_{ij}]\tilde{x}_j \leq \tilde{x}_i + \sum_{j \in J_N} \bar{a}_{ij}\tilde{x}_j = \bar{b}_i \quad i \in J_B, \tag{2.2}$$

comme la partie gauche de (2.2) est à valeurs entières, alors :

$$\tilde{x}_i + \sum_{j \in J_N} [\bar{a}_{ij}]\tilde{x}_j \leq [\bar{b}_i], \quad i \in J_B,$$

comme la formule (2.1) est valable pour toute solution réalisable, il s'ensuit que :

$$\{\bar{b}_i\} - \sum_{j \in J_N} \{\bar{a}_{ij}\}\tilde{x}_j \leq 0, \quad i \in J_B,$$

d'où l'expression de la *coupe de Gomory* :

$$\boxed{-\sum_{j \in J_N} \{\bar{a}_{ij}\}x_j \leq -\{\bar{b}_i\}, \quad (i \in J_B).}$$

Avec cette coupe, on n'élimine pas de solution réalisable à valeurs entières de l'ensemble des solutions réalisables.

Schéma de l'algorithme des coupes de Gomory

ENTRÉE : Une instance d'un problème de PLE ;

SORTIE : Une solution optimale x^*.

Étape 1 : Résoudre le PL relaxé par la méthode du simplexe.

Aller à (2).

Étape 2 : **Si** la solution optimale est entière, alors :

Terminer, et cette solution est la solution optimale du problème initial.

Sinon

Aller à (3).

Étape 3 : Sélectionner une ligne comme ligne génératrice à partir du tableau de simplexe.

Ajouter au système d'équations obtenu à la dernière itération la coupe :

$$-\sum_{j\in J_N}\{\bar{a}_{ij}\}x_j \leq -\{\bar{b}_i\}, i \in J_B.$$

Aller à (4).

Étape 4 : Appliquer l'algorithme dual du simplexe à partir du tableau optimal précédent.

retourner à (2).

Exemple 2.6.2

Considérons le programme linéaire en nombres entiers suivant :

$$(PLE)\begin{cases} max\ \ z(x) = 3x_1 + 2x_2 + x_3 + 2x_4, \\ s.c \quad x_1 - x_2 + 2x_3 + x_4 \leq 11, \\ \qquad\quad x_2 + x_3 + x_4 \leq 7, \\ \quad 3x_1 \qquad - x_3 - 3x_4 \leq 5, \\ \quad x_i \geq 0, \quad i = \overline{1.4}, \quad x \text{ entier.} \end{cases} \Rightarrow (PLRS)\begin{cases} max\ \ z(x) = 3x_1 + 2x_2 + x_3 + 2x_4, \\ s.c \quad x_1 - x_2 + 2x_3 + x_4 + x_5 = 11, \\ \qquad\quad x_2 + x_3 + x_4 + x_6 = 7, \\ \quad 3x_1 \qquad - x_3 - 3x_4 + x_7 = 5, \\ \quad x_i \geq 0, \quad i = \overline{1.7}. \end{cases}$$

Après la résolution du (PLRS), le dernier tableau du simplexe donne :

		c	3	2	1	2	0	0	0
c_B	x_B	b	x_1	x_2	x_3	x_4	x_5	x_6	x_7
2	x_4	49/9	0	0	10/9	1	1/3	1/3	$-1/9$
2	x_2	14/9	0	1	$-1/9$	0	$-1/3$	2/3	1/9
3	x_1	64/9	1	0	7/9	0	1/3	1/3	2/9
$z = 106/3$		E_j	0	0	10/3	0	1	3	2/3

La solution du (PLRS) est $x = (64/9, 14/9, 0, 49/9, 0, 0, 0)' \notin \mathbb{N}^4$ avec $z(x) = 106/3$, donc nous générons une coupe par rapport à x_2, nous obtenons la coupe suivante :

$$\frac{-8}{9}x_3 - \frac{2}{3}x_5 - \frac{2}{3}x_6 - \frac{1}{9}x_7 + x_8 = \frac{-5}{9}.$$

En ajoutant cette coupe et en réoptimisant, nous obtenons la solution optimale suivante :
$x = (41/6, 11/6, 0, 31/6, 0, 0, 0)'$ avec $z(x) = 69/2$, cette solution n'est pas entière nous générons une nouvelle coupe par rapport à x_1, nous aurons la coupe suivante :

$$\frac{-1}{3}x_1 - \frac{1}{6}x_7 - \frac{1}{2}x_8 + x_9 = \frac{-5}{6}.$$

En ajoutant cette coupe et en réoptimisant, nous obtenons la solution optimale suivante :
$x = (6, 1, 0, 6)'$ qui est entière, donc optimale pour le programme original en nombres entiers, avec $z(x) = 32$.

2.6.2 Méthode de séparation et d'évaluation

2.6.2.1 Présentation de la méthode

Nous avons présenté une méthode de résolution des problèmes de PLE en utilisant des coupes, notamment celle de Gomory. Cette dernière entame le problème en gardant le domaine réalisable convexe, ce qui peut être inefficace dans certaines configurations. Ainsi, on peut résoudre les problèmes de PLE en fractionnant le domaine en régions faisables bornées par des frontières alignées sur des entiers : on sépare ainsi les domaines, et on évalue quelle région explorer en premier. Cette méthode est appelée recherche arborescente par séparation et évaluation, on l'appelle aussi méthode de *Branch-and-Bound*.

Le premier algorithme de cette méthode a été introduit par Land et Doig [15]."*Banch-and-Bound*" est le terme très suggestif donné par Little et al. [16] à leur méthode pour

résoudre le problème du voyageur de commerce.

L'article de 1960 par Ailsa Land et Alison Doig a présenté cette méthode qui a été si importante pour obtenir des solutions aux problèmes de PLE. En fait, la plupart des codes informatiques modernes réussissent grâce aux méthodes de branchement en s'inspirant de ces différentes méthodes telles que *Branch and Cut*, *Branch and Price*, *Branch and Cut and Price...*

La procédure arborescente s'applique à plusieurs problèmes, citons :

- les problèmes de recouvrement minimum sur un graphe ;
- les recherches d'ensembles intérieurement stables maximaux et extérieurement stables minimaux ;
- les problèmes du type "voyageur de commerce", et plus généralement les problèmes de tournées ;
- les problèmes d'ordonnancement à contraintes disjonctives ;
- les programmes linéaires à variables entières ou mixtes.

Soit le problème d'optimisation combinatoire suivant :

$$\max_{x \in S} \; z(x), \tag{2.3}$$

où $|S|$ est fini. On cherche à résoudre ce problème en partitionnant S en sous-ensembles ; on tente de réduire au maximum le nombre de sous-ensembles à examiner lors de la recherche d'une solution optimale.

a) Énumération des solutions [17]

On présente les sous-ensembles S_i que l'on considère pour des sommets s_i ($S_0 = S$). Une séparation de S_i est une opération qui consiste à remplacer S_i par une collection S_i^* de sous-ensembles $S_{i_1}, S_{i_2}, ..., S_{i_k}$ de S_i tels que $S_{i_1} \cup S_{i_2} \cup ... \cup S_{i_k} = S_i$ (les S_{i_k} sont en général disjoints et $k < \infty$). On relie le sommet s_i à chacun des s_{i_k} par un arc (s_i, s_{i_k}). On obtient finalement une arborescence de racine s_0.

Propriété 1 *S'il y a un chemin de s_i à s_j alors $S_i \supseteq S_j$.*

Un sommet s_j (ou un ensemble S_j) est *sondé* (coupé) s'il n'y a plus d'intérêt à le séparer ; il est *actif* dans le cas contraire.

b) Principe des méthodes de séparation et évaluation

En partant d'un sommet actif, on cherche soit à le séparer soit à le sonder ; on parcourt ainsi l'arborescence ; si le sommet où l'on se trouve est sondé, on retourne à l'unique sommet précédent dans l'arborescence et l'on continue jusqu'à ce que l'on trouve un sommet ayant au-moins un successeur actif ; lorsqu'il n'y a plus de sommets actifs, l'énumération est alors terminée.

c) Choix d'une stratégie d'exploration des nœuds [17] [18]

L'exploration des nœuds de l'arborescence des solutions obéit à une stratégie donnée. De nombreuses stratégies ont pu être proposées, dont aucune ne s'avère systématiquement meilleure.

Dans la littérature, on distingue trois types de stratégies : **Profondeur d'abord**, **largeur d'abord** et **meilleur d'abord**.

Dans la stratégie *profondeur d'abord*, on choisit le sommet de niveau le plus élevé parmi les sommets non encore séparés. S'il en existe plusieurs, on pourra choisir celui qui correspond à l'évaluation la plus faible (cas de maximisation). Cette méthode a pour but de mettre en évidence le plus tôt possible une solution du problème.

Dans la stratégie *largeur d'abord*, elle est encore appelée méthode SEP (Séparation et Évaluation Progressive) on choisit systématiquement le sommet ayant l'évaluation la plus faible (cas de maximisation), en remarquant celui qui a le plus de chances de contenir une solution optimale parmi ses successeurs. Le risque, avec cette méthode, est d'avoir à explorer une fraction importante de l'arborescence avant de découvrir une solution. En la comparant avec la précédente, la solution trouvée ici est généralement de meilleure qualité (c'est-à-dire de coût moins élevé).

Dans le cas des deux stratégies précédentes, l'ordre d'exploration est défini au départ. Par contre, dans une troisième stratégie appelée *meilleur d'abord*, il ne l'est pas. On sépare le nœud ayant la meilleure valeur de la fonction objectif.

Dans la littérature, on trouve aussi d'autres stratégies d'exploration, comme par exemple, choisir le nœud qui possède la somme des parties fractionnaires la plus élevée, cette dernière est calculée comme suit : $f = \sum_{j=1}^{n} \min(\{x_j\}, 1 - \{x_j\})$.

d) Sondage à l'aide de bornes

Soit le sous-problème

$$\max_{x \in S_j} \; z(x),$$

associé au sommet s_j ; on obtient une borne supérieure $\overline{z}_j$ de la valeur optimale de z sur S_j en remplaçant le sous-problème par une relaxation

$$\max_{x \in T_j \supseteq S_j} \; z(x) \qquad (\overline{z}_j = -\infty, \;\; T_j = \emptyset). \tag{2.4}$$

On obtient une borne inférieure $\underline{z}_j$ en choisissant un $x' \in S_j$ et en posant $\underline{z}_j = z(x')$.

Ainsi, un sommet s_j (associé à S_j) sera sondé si :

1. $\overline{z}_j = \underline{z}_j$, c'est-à-dire si on a obtenu une solution optimale $x^0(s_j)$ pour la relaxation qui est telle que $x^0(s_j) \in S_j$;
2. $\overline{z}_j \leqslant \underline{z}_0$, c'est-à-dire que dans S_j il ne peut exister de solution qui donne à z une valeur meilleure que $\underline{z}_0$, la meilleure valeur obtenue jusqu'à présent (initialement, on peut poser $\underline{z}_0 = -\infty$).

2.6.2.2 Application de *Branch-and-Bound* à la PLE

L'idée de départ consiste à représenter l'ensemble des solutions du PLE à résoudre par une arborescence et à parcourir cette arborescence en utilisant le mécanisme d'évaluation pour éviter l'examen effectif de sous-arborescences ne pouvant contenir de solution optimale. Dans ce type de méthode, l'efficacité algorithmique dépend de la qualité de l'évaluation en chaque nœud de l'arborescence : Plus faible est l'écart entre la valeur fournie par le mécanisme d'évaluation et la valeur exacte de l'optimum entier, plus la proportion des nœuds à examiner explicitement est réduite [19].

Posons

$$S = \{x \in \mathbb{R}^n : Ax = b, x \geq 0, \;\; x \text{ entier}\}.$$

Soit $A^j x = b^j$ un système d'équations obtenues en rajoutant quelques contraintes au système initial ; soit alors :

$$S_j = \{x | A^j x = b^j, x \geq 0, \;\; x \text{ entier}\} \subseteq S. \tag{2.5}$$

Pour la relaxation, on utilise :

$$T_j = \{x | A^j x = b^j, x \geq 0 \ \} \supseteq S_j. \tag{2.6}$$

On obtient donc un PL ordinaire.

Pour la séparation, si x_{b_i} a une valeur y_{i_0} non entière à l'optimum du PL, on peut prendre :

$$S_j^* = \{S_j \cup \{x| \ x_{b_i} \leqslant [y_{i_0}]\}, S_j \cup \{x| \ x_{b_i} \geqslant [y_{i_0}] + 1\}\}. \tag{2.7}$$

En résumé, la méthode *Branch-and-Bound* appartient à la famille des méthodes de recherche arborescente. Son principe est de diviser (séparer) l'espace de recherche en plusieurs sous-ensembles, évaluer la borne supérieure (cas de maximisation) de la solution optimale du sous-problème, en résolvant la relaxation linéaire, et la comparer au coût de la solution réalisable courante du problème, appelé "référence". Ce processus est réitéré pour les sous-problèmes dont la borne supérieure est supérieure à la référence, c'est-à-dire les sous-problèmes susceptibles de contenir une meilleure solution. Le schéma dans la figure ci-après résume cette méthode.

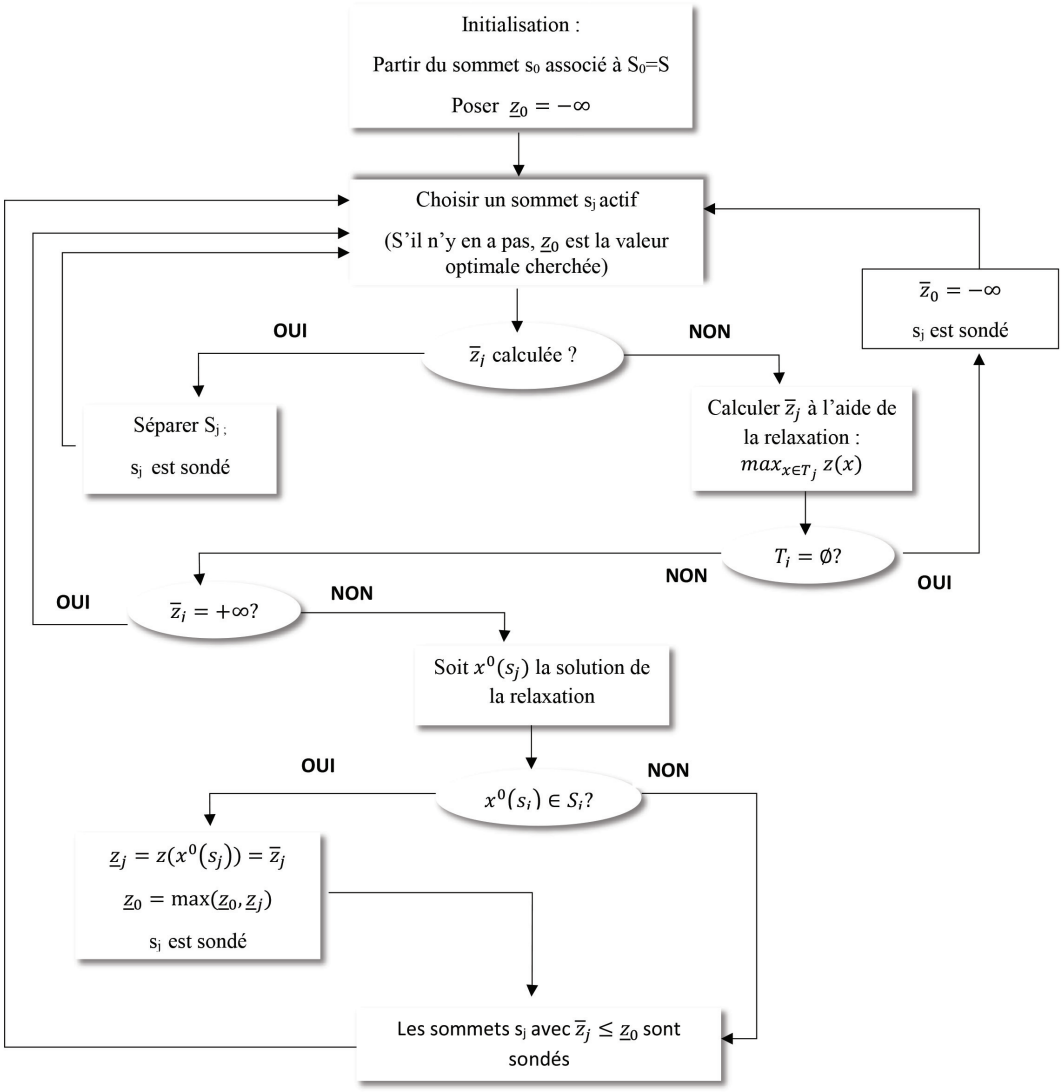

FIGURE 2.1 – Schéma de l'algorithme de *Branch-and-Bound*

Exemple 2.6.3

$$(PLE)\begin{cases} max\ \ z(x) = 3x_1 + 2x_2 + x_3 + 2x_4, \\ s.c \quad x_1 - x_2 + 2x_3 + x_4 \leq 11, \\ \qquad x_2 + x_3 + x_4 \leq 7, \\ \quad 3x_1 \quad - x_3 - 3x_4 \leq 5, \\ \quad x_i \geq 0, \quad i = \overline{1.4}, \quad x \text{ entier}. \end{cases} \Rightarrow (PLR)\begin{cases} max\ \ z(x) = 3x_1 + 2x_2 + x_3 + 2x_4, \\ s.c \quad x_1 - x_2 + 2x_3 + x_4 + x_5 = 11, \\ \qquad x_2 + x_3 + x_4 + x_6 = 7, \\ \quad 3x_1 \quad - x_3 - 3x_4 + x_7 = 5, \\ \quad x_i \geq 0, \quad i = \overline{1.7}. \end{cases}$$

Soient $S_0 = S = \{x |\ Ax = b, x \geqslant 0 \text{ entier}\}$ et $\underline{z}_0 = -\infty$.

On va calculer $\overline{z}_0$ à l'aide de la relaxation :

$$T_0 = \{x |\ Ax = b,\ x \geqslant 0\}.$$

Après la résolution avec la méthode de simplexe on obtient la solution optimale suivante :

$x^0 = (64/9, 14/9, 0, 49/9)' \notin S^0$ et $\bar{z}_0 = 106/9$; s_0 est actif ; on le sépare :

$$S_0^* = \{\underbrace{S_0 \cup \{x|\ x_2 \geq 2\}}_{S_1}, \underbrace{S_0 \cup \{x|\ x_2 \leq 1\}}_{S_2}\}.$$

$$T_1 = \{x|\ Ax = b,\ x_2 \geqslant 2, x_1, x_2, x_3 \geqslant 0\}.$$

En ajoutant la contrainte suivante : $\frac{-1}{9}x_3 - \frac{1}{3}x_5 + \frac{2}{3}x_6 + \frac{1}{9}x_7 + x_8 = \frac{-4}{9}$, et en utilisant le dual du simplexe on aura la solution optimale du (PLR) suivante : $x = (20/3, 2, 0, 5)' \notin S^0$ et $\bar{z}_0 = 34$; s_1 est actif ; on le sépare :

$$S_1^* = \{\underbrace{S_1 \cup \{x|\ x_1 \geq 7\}}_{S_3}, \underbrace{S_1 \cup \{x|\ x_1 \leq 6\}}_{S_4}\}.$$

$$T_{13} = \{x|\ Ax = b,\ x_2 \geqslant 2,\ x_1 \geq 7,\ x_1, x_2, x_3 \geqslant 0\}.$$

Ce problème n'admet pas de solution. $T_{13} = \emptyset, \overline{z}_3 = -\infty$; s_3 est sondé.
On choisit s_4 qui est actif ; on calcule $\overline{z}_4$:

$$T_{14} = \{x|\ Ax = b,\ x_2 \geqslant 2,\ x_1 \leq 6,\ x_1, x_2, x_3 \geqslant 0\}.$$

On aura la nouvelle contrainte : $\frac{-6}{9}x_3 - x_6 - \frac{1}{3}x_7 - x_8 + x_9 = \frac{-2}{3}$. Ainsi, la solution optimale suivante : $x = (6, 2, 0, 5)'$ qui est entière et $\bar{z}_4 = 32$, alors on coupe le sommet s_4.

$$T_2 = \{x|\ Ax = b,\ x_2 \leq 1,\ x_1, x_2, x_3 \geqslant 0\}.$$

En ajoutant la contrainte suivante : $\frac{1}{9}x_3 + \frac{1}{3}x_5 - \frac{2}{3}x_6 - \frac{1}{9}x_7 + x_{10} = \frac{-5}{9}$. Ainsi, la solution optimale suivante : $x = (41/6, 1, 0, 31/6)' \notin S^0$ et $\bar{z}_0 = 197/6$; s_2 est actif on le sépare :

$$S_2^* = \{\underbrace{S_2 \cup \{x|\ x_1 \geq 7\}}_{S_5}, \underbrace{S_2 \cup \{x|\ x_1 \leq 6\}}_{S_6}\}.$$

$$T_{25} = \{x|\ Ax = b,\ x_2 \leq 1,\ x_1 \geq 7,\ x_1, x_2, x_3 \geqslant 0\}.$$

Ce problème n'admet pas de solution. $T_{25} = \emptyset, \overline{z}_5 = -\infty$; s_5 est sondé. On choisit s_6 qui est actif ; on calcule $\overline{z}_6$:

$$T_{26} = \{x|\ Ax = b,\ x_2 \leq 1,\ x_1 \leq 6,\ x_1, x_2, x_3 \geqslant 0\}.$$

On aura la nouvelle contrainte : $\frac{-5}{6}x_3 - \frac{1}{2}x_5 - \frac{1}{6}x_7 - \frac{1}{2}x_{10} + x_{11} = \frac{-5}{6}$. Ainsi, la solution optimale suivante : $x = (6, 1, 0, 6)'$ et elle est entière, avec $\bar{z}_6 = 32$, alors on coupe le sommet s_6.
Il ne reste plus de sommets actifs ; on a deux solutions optimales : $x^0 = (6, 2, 0, 5)'$ et $x^1 = (6, 1, 0, 6)'$ et $z = 32$.

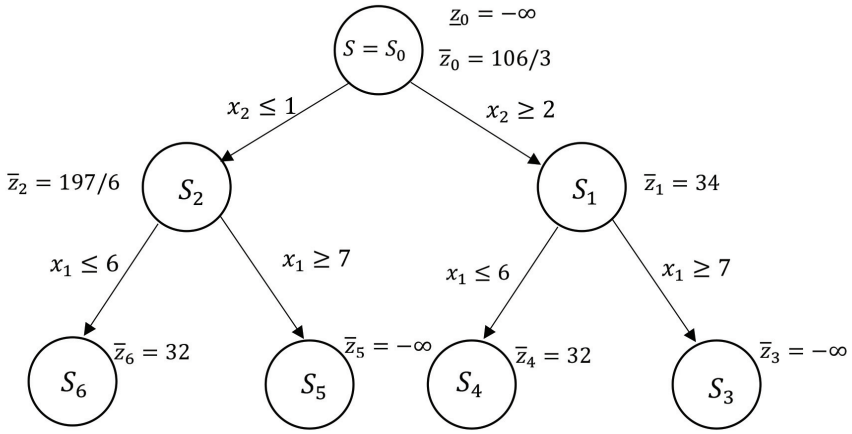

FIGURE 2.2 – Représentation sous forme d'arborescence

2.6.3 Programmation dynamique

2.6.3.1 Présentation de la méthode

La programmation dynamique est une technique permettant de résoudre des problèmes combinatoires et autres. Elle s'utilise en avenir certain (déterministe) ou en avenir incertain, correspondant à un processus aléatoire ou stochastique. Elle est fondée sur le principe d'optimalité de Bellman, qui permet de définir une solution optimale par succession de choix séquentiels optimaux.

Un problème de ce type est représenté de la façon suivante :

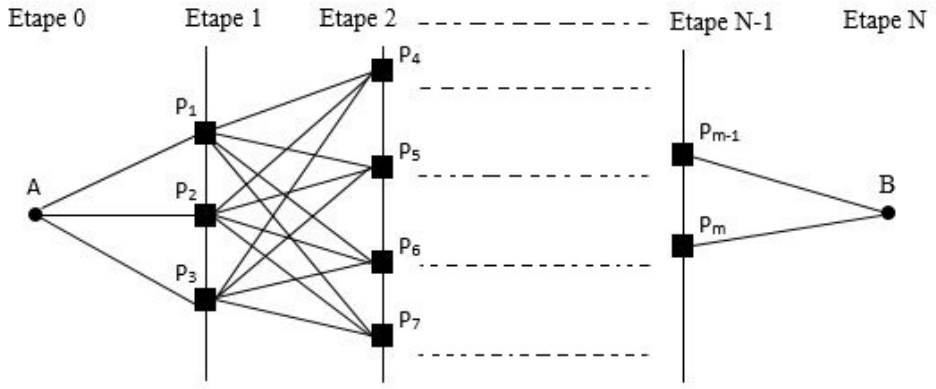

FIGURE 2.3 – Schéma représentatif de la programmation dynamique

Le départ figure au point **A** et l'objectif à atteindre est au point **B**.

Le passage de chacune des **N** étapes est obligatoire. Sur l'itinéraire, il est possible de choisir m points de passage $P_1, P_2, ..., P_m$.

En partant du point **A**, il est possible de passer à l'étape 1 par le point P_1, P_2 ou P_3, puis à partir de ce point de passer à l'étape 2 par le point P_4, P_5, P_6 ou encore P_7, et ainsi de suite jusqu'au point d'arrivée **B**.

Une série de décisions s'effectue alors entre chaque transition. En effet, la programmation dynamique permet de résoudre des problèmes caractérisés par des décisions interdépendantes et séquentielles.

Cette théorie s'applique dans des domaines divers, comme par exemple la recherche d'une stratégie commerciale permettant de déterminer des prix de vente de produits ou encore en stratégie et politique financières, pour effectuer des choix d'investissement. En logistique, les politiques et stratégies de gestion des stocks, de recherche d'itinéraires de transport ou encore de gestion optimale des tournées de livraison sont d'autres champs d'application. Cet outil très polyvalent s'utilise aussi en gestion de production et pour définir des politiques ou stratégies d'achats [20].

2.6.3.2 Programmation dynamique pour la PLE

Principe de Bellman [21] : Ce principe est basé sur une technique "d'optimisation arrière" qui consiste à trouver une formulation récursive du problème donné. Ensuite, en procédant à un découpage étape par étape pour obtenir la formule de récurrence.

Voici un problème de PLE :

$$\begin{cases} z = max \sum_{j=1}^{n} c_j x_j, \\ \sum_{j=1}^{n} a_j x_j \leq b, \\ x_j \geq 0, \quad x_j \text{ entiers}, \ j = 1, ..., n. \end{cases}$$

Soit $\beta_k(y)$ la valeur du sous-problème défini par les k premières variables et soit $b = y$.

$$\beta_k(y) = \max\{\sum_{j=1}^{k} c_j x_j | \sum_{j=1}^{k} a_j x_j \leq y, x_j \geq 0, x_j \in \mathbb{N}, j = 1, ..., k\} \quad ...(*)$$

Si $k \geq 2$, et pour $y = 0, 1, ..., b$, on écrit la formule $(*)$ sous cette forme :

$$\beta_k(y) = \max_{x_k=0,1,...,[y/a_k]} c_k x_k + \max\{\sum_{j=1}^{k-1} c_j x_j | \sum_{j=1}^{k-1} a_j x_j \leq y, x_j \geq 0, x_j \in \mathbb{N}, j = 1, ..., k-1\}.$$

Cette expression est équivalente à :

$$\beta_k(y) = \max_{x_k=0,1,...,[y/a_k]} \{c_k x_k + \beta_{k-1}(y - a_k x_k)\}.$$

2.6.4 Méthode directe de support pour la résolution d'un problème de PL à variables bornées

Elle a été développée par R. Gabassov et F. M. Kirillova dans les années 70. Elle a la particularité de tenir compte des spécificités des problèmes tels qu'ils sont formulés lors de leur modélisation première. Les algorithmes qui sont déduits de cette méthode utilisent au maximum la structure particulière des problèmes pour plus d'efficacité.

2.6.4.1 Position du problème

On se basant sur les cours [23], considérons le problème de PLE et à variables bornées, s'écrivant sous la forme canonique suivante :

$$(PLE) \begin{cases} max\ z(x) = c'x, \\ s.c \quad Ax = b, \\ \qquad d^- \leq x \leq d^+, x \quad \text{entier}, \end{cases} \tag{2.8}$$

où A est une matrice rationnelle d'ordre $m \times n$, $rangA = m < n$; b est un vecteur rationnel de dimension m ; c, x, d^- et d^+ sont des vecteurs de dimension n dans $\mathbb{Z}^n$.
On considère alors le problème relaxé associé au problème (2.8) suivant :

$$(PLR) \begin{cases} max\ z(x) = c'x, \\ s.c \quad Ax = b, \\ \qquad d^- \leq x \leq d^+. \end{cases} \tag{2.9}$$

Notons :

$I = \{1, 2, ..., m\}$: l'ensemble d'indices des lignes de A ;

$J = \{1, 2, ..., n\}$: l'ensemble d'indices des colonnes de A.

Donnons les définitions suivantes :

- Un vecteur x vérifiant les contraintes $Ax = b, \quad d^- \leq x \leq d^+$, x entier, est appelé solution réalisable du problème (2.8). L'ensemble des solutions réalisables est alors donné par :

$$S = \{x \in \mathbb{Z}_+^n : \quad Ax = b, \quad d^- \leq x \leq d^+\}.$$

  De même, on définit par $X = \{x \in \mathbb{R}^n : \quad Ax = b, \quad d^- \leq x \leq d^+\}$, l'ensemble des solutions réalisables du problème (2.9).
- Une solution réalisable x^0 est dite optimale pour le problème (2.9) si :

$$z(x^0) = \max_{x \in X} \quad c'x.$$

- Une solution réalisable x^ε est appelée ε-optimale ou suboptimale si

$$z\ (x^0) - \ z\ (x^\varepsilon) = c'x^0 - c'x^\varepsilon \leq \varepsilon,$$

  où x^0 est une solution optimale du problème (2.9) et $\varepsilon \geq 0$ une précision choisie à l'avance .
- Soit un sous-ensemble d'indice $J_B \subset J$ tel que $J = J_B \cup J_N$, $J = J_B \cap J_N = \emptyset$, $\mid J_B \mid = m$.
- En vertu de la partition de $J = J_B \cup J_N$, on peut alors écrire et fractionner les vecteurs et les matrices de la manière suivante : $x = \begin{pmatrix} x_B \\ x_N \end{pmatrix}$, $x_B = (x_j, j \in J_B)$, $x_N = (x_j, j \in J_N)$; $c = \begin{pmatrix} c_B \\ c_N \end{pmatrix}$, $c_B = (c_j, j \in J_B)$, $c_N = (c_j, j \in J_N)$;

  $A(I, J) = (a_{ij},\ i \in J\ ,\ j \in J) = (a_j,\ j \in J)$, $a_j = \begin{pmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{pmatrix}$ est la $j^{ème}$ colonne de la matrice A ;

  $A = A(I,\ J) = (A_B \mid A_N)$, $A_B = A(I,\ J_B)$, $A_N = A(I,\ J_N)$.

  L'ensemble J_B est alors appelé support si :

$$detA_B = detA(I, J_B) \neq 0.$$

– Le couple $\{x, J_B\}$ formé du plan x et du support J_B est appelé plan de support. Un plan de support $\{x, J_B\}$ est dit basique si

$$x_j = d_j^- \vee d_j^+, \forall j \in J_N.$$

– Le plan de support $\{x, J_B\}$ est dit non dégénéré si :

$$d_j^- < x_j < d_j^+, \quad \forall j \in J_B.$$

2.6.4.2 Formule d'accroissement de la fonction objectif

Soit $\{x, J_B\}$ un plan de support du problème (2.9). On considère une autre solution réalisable quelconque $\bar{x} = x + \Delta x$. On a donc :

$$\Delta z = z(\bar{x}) - z(x) = c'\bar{x} - c'x = c'\Delta x. \tag{2.10}$$

Par ailleurs on a :

$$A\bar{x} = Ax = b \Rightarrow A\Delta x = 0 \Rightarrow A\Delta x = A_B\Delta x_B + A_N\Delta x_N = 0.$$

D'où :

$$\Delta x_B = -A_B^{-1}A_N\Delta x_N. \tag{2.11}$$

Ainsi, l'accroissement (2.10) devient :

$$\Delta z = c'\Delta x = -c'_B A_B^{-1}A_N\Delta x_N + c'_N\Delta x_N = -(c'_B A_B^{-1}A_N - c'_N)\Delta x_N. \tag{2.12}$$

On définit le vecteur des potentiels u ainsi que le vecteur des estimations E comme suit :

$$u' = c'_B A_B^{-1} \text{ et } E' = u'A - c' \Leftrightarrow E_j = u'a_j - c_j, \quad j \in J,$$

où $E' = (E'_B, E'_N)$, avec :

$$E'_B = u'A_B - c'_B = c'_B A_B^{-1}A_B - c'_B = 0,$$

$$E'_N = u'A_N - c'_N = c'_B A_B^{-1}A_N - c'_N.$$

Finalement, la formule d'accroissement (2.12) prend la forme finale suivante :

$$\Delta z = -E'_N\Delta x_N = -\sum_{j \in J_N} E_j(\bar{x}_j - x_j). \tag{2.13}$$

2.6.4.3 Critère d'optimalité

On a le théorème d'optimalité suivant :

Théorème 2.6.2 *Soit $\{x, J_B\}$ un plan de support du problème (2.9). Les relations :*

$$\begin{cases} E_j \geq 0, & pour \quad x_j = d_j^-, \\ E_j \leq 0, & pour \quad x_j = d_j^+, \\ E_j = 0, & pour \quad d_j^- < x_j < d_j^+, \qquad j \in J_N, \end{cases} \tag{2.14}$$

sont suffisantes pour l'optimalité du plan de support $\{x, J_B\}$, et aussi nécessaire si la solution réalisable du support est non-dégénérée.

Preuve 2.6.1 (*Condition suffisante*) *Soit $\{x, J_B\}$ un plan de support vérifiant les relations (2.14). Pour tout plan de support $\bar{x}$ du problème (2.9), la formule d'accroissement (2.13)donne :*

$$z(\bar{x}) - z(x) = -\sum_{j \in J_N} E_j(\bar{x}_j - x_j) = -\sum_{E_j > 0,\ j \in J_N} E_j(\bar{x}_j - d_j^-) - \sum_{E_j < 0,\ j \in J_N} E_j(\bar{x}_j - d_j^+),$$

où

$$d_j^- \leq \bar{x}_j \leq d_j^+ \Rightarrow \bar{x}_j - d_j^- \geq 0 \ et \ \bar{x}_j - d_j^+ \leq 0,$$

donc

$$z(\bar{x}) - z(x) \leq 0 \Rightarrow z(\bar{x}) \leq z(x), \ \forall \ \bar{x} \ solution \ réalisable.$$

Par conséquent, le vecteur x est une solution optimale du problème (2.9).

Condition Nécessaire

Soit $\{x, J_B\}$ un plan de support optimal non dégénéré du problème (2.9), et supposons que les relations d'optimalité (2.14) ne sont pas vérifiées, c'est-à-dire $\exists j_0 \in J_N$ tel que :

$$E_{j_0} < 0 \ et \ x_{j_0} < d_{j_0}^+ \ ou \ bien \ E_{j_0} > 0 \ et \ x_{j_0} > d_{j_0}^-.$$

On construit alors une autre solution réalisable $\bar{x} = x + \theta l$, *où* $\theta > 0$ *(nombre réel) et* $l = \{l_j,\ j \in J\}$ *un vecteur de direction construit comme suit :*

$$\begin{cases} l_{j_0} = -sign\ E_{j_0}, \\ l_j = 0, \quad j \neq j_0, \quad j \in J_N, \\ l_B = l(J_B) = -A_B^{-1} A_N l_N = A_B^{-1} a_{j_0} sign\ E_{j_0}. \end{cases} \tag{2.15}$$

Le vecteur $\bar{x}$ *vérifie les contraintes principales* $A\bar{x} = b$. *Pour que* $\bar{x}$ *soit une solution réalisable du problème (2.9), il doit aussi vérifier l'inégalité :*

$$d^- \leq \bar{x} \leq d^+ \Longleftrightarrow d^- \leq x + \theta l \leq d^+ \Longleftrightarrow d^- - x \leq \theta l \leq d^+ - x.$$

c'est-à-dire :

$$\begin{cases} d_j^- - x_j \leq \theta l_j \leq d_j^+ - x_j, \ \ j \in J_B, \\ d_{j_0}^- - x_{j_0} \leq -\theta sign\ E_{j_0} \leq d_{j_0}^+ - x_{j_0}. \end{cases} \tag{2.16}$$

Puisque le plan de support $\{x, J_B\}$ *est non dégénéré, on a alors :*

$$d_j^- - x_j < 0 < d_j^+ - x_j, \ \ j \in J_B,$$

et

$$\begin{cases} d_{j_0}^- - x_{j_0} < 0 \quad si \quad E_{j_0} > 0, \\ d_{j_0}^+ - x_{j_0} > 0 \quad si \quad E_{j_0} < 0. \end{cases}$$

Par conséquent pour un nombre positif θ *assez petit, le vecteur* $\bar{x} = x + \theta l$ *est une solution réalisable du problème (2.9) et l'accroissement (2.13) nous donne :*

$$\Delta z = -\sum_{j \in J_N} E_j(\bar{x}_j - x_j) = -\theta \sum_{j \in J_N} E_j l_j = \theta E_{j_0} sign\ E_{j_0} = \theta \mid E_{j_0} \mid > 0, \tag{2.17}$$

c'est-à-dire $z(\bar{x}) - z(x) > 0 \Rightarrow z(\bar{x}) > z(x)$, *ce qui constitue une contradiction avec le fait que* x *est une solution optimale du problème (2.9).*

Par conséquent, les relations (2.14) sont nécessairement vérifiées si $\{x, J_B\}$ *est un plan optimal du problème (2.9).*

2.6.4.4 Estimation de suboptimalité

Pour estimer l'écart qui existe entre la valeur optimale $z(x^0)$ et une autre valeur $z(x)$ d'un plan de support quelconque $\{x, J_B\}$, on remplace le vecteur $\bar{x}$ par x^0 dans l'accroissement (2.13) et on aura :

$$z(x^0) - z(x) = -\sum_{j \in J_N} E_j(x_j^0 - x_j) = \sum_{E_j > 0, j \in J_N} E_j(x_j - x_j^0) + \sum_{E_j < 0, j \in J_N} E_j(x_j - x_j^0). \quad (2.18)$$

On a

$$d_j^- \leq x_j^0 \leq d_j^+, \quad j \in J.$$

D'où

$$x_j - x_j^0 \leq x_j - d_j^- \text{ et } x_j - x_j^0 \geq x_j - d_j^+.$$

Donc

$$\begin{cases} E_j(x_j - x_j^0) \leq E_j(x_j - d_j^-), & si \quad E_j > 0, \\ E_j(x_j - x_j^0) \leq E_j(x_j - d_j^+), & si \quad E_j < 0. \end{cases}$$

Ainsi, on obtient :

$$z(x^0) - z(x) \leq \sum_{E_j > 0, j \in J_N} E_j(x_j - d_j^-) + \sum_{E_j < 0, j \in J_N} E_j(x_j - d_j^+). \quad (2.19)$$

Le nombre

$$\beta(x, J_B) = \sum_{E_j > 0, j \in J_N} E_j(x_j - d_j^-) + \sum_{E_j < 0, j \in J_N} E_j(x_j - d_j^+), \quad (2.20)$$

est appelé *estimation de suboptimalité* du plan de support $\{x, J_B\}$.

Théorème 2.6.3 *Soit $\{x, J_B\}$ un plan de support du problème (2.9) et $\varepsilon \geq 0$. Si $\beta(x, J_B) \leq \varepsilon$ alors x est ε-optimal.*

Preuve 2.6.2 *En vertu de (2.20), on obtient :*

$$z(x^0) - z(x) \leq \beta(x, J_B) \leq \varepsilon \Longrightarrow x \text{ est } \varepsilon\text{-optimal.}$$

2.6.4.5 Algorithme de résolution

Soit $\varepsilon \geq 0$ un nombre réel quelconque et un plan de support $\{x, J_B\}$. Une itération de l'algorithme consiste à faire le passage de $\{x, J_B\}$ vers $\{\bar{x}, \bar{J}_B\}$ tel que $z(\bar{x}) \geq z(x)$. Pour cela, on construit une nouvelle solution réalisable $\bar{x}$ de la manière suivante :

$$\bar{x} = x + \theta l, \qquad \theta \geq 0,$$

où l une direction d'amélioration, θ est le pas le long de cette direction.
Pour que l'accroissement (2.19) soit maximal, il faut prendre θ aussi grand que possible et choisir l'indice j_0 tel que :

$$\mid E_{j_0} \mid = \max_{j \in J_{NNO}} \mid E_j \mid,$$

où J_{NNO} représente le sous-ensemble des indices non basiques non optimaux, tel que :

$$J_{NNO} = \{j \in J_N : [E_j > 0 \quad \wedge \quad x_j > d_j^-] \vee [E_j < 0 \quad \wedge \quad x_j < d_j^+]\}.$$

D'autre part, le pas θ doit vérifier les relations (2.16) et la direction l est calculée suivant les relations (2.15).

On calcule les différentes valeurs maximales du pas θ : pour les indices $j \in J_B$, on a la formule suivante :

$$\theta_j = \begin{cases} \dfrac{d_j^+ - x_j}{l_j}, & l_j > 0, \\ \dfrac{d_j^- - x_j}{l_j}, & l_j < 0, \\ \infty, & l_j = 0. \end{cases}$$

Pour les θ_j basiques, le pas maximal θ qui vérifie les relations (2.16), se calcule comme suit :

$$\theta = \theta_{j_1} = \min_{j \in J_B} \theta_j.$$

Pour l'indice j_0, on obtient :

$$\theta_{j_0} = \begin{cases} d_{j_0}^+ - x_{j_0}, & si \quad l_{j_0} = 1, \\ x_{j_0} - d_{j_0}^-, & si \quad l_{j_0} = -1. \end{cases}$$

Le pas maximal θ^0 est donc :

$$\theta^0 = \min\{\theta_{j_1}, \theta_{j_0}\}, \quad j_1 \in J_B, \quad j_0 \in J_N. \tag{2.21}$$

Donc la nouvelle solution réalisable est : $\bar{x} = x + \theta^0 l$.

Calculons l'estimation de suboptimalité pour le plan $\{\bar{x}, J_B\}$:

$$\beta(\bar{x}, J_B) = \sum_{E_j>0,\ j\in J_N} E_j(\bar{x}_j - d_j^-) + \sum_{E_j<0,\ j\in J_N} E_j(\bar{x}_j - d_j^+).$$

On a

$$\bar{x}_j = x_j, \quad si \quad j \in J_N \quad et \quad j \neq j_0, \qquad \bar{x}_{j_0} = \begin{cases} x_{j_0} - \theta^0, & si \quad E_{j_0} > 0, \\ x_{j_0} + \theta^0, & si \quad E_{j_0} < 0. \end{cases}$$

En remplaçant, on aura :

$$\beta(\bar{x}, J_B) = \beta(x, J_B) - \theta^0 \mid E_{j_0} \mid . \tag{2.22}$$

Si $\beta(\bar{x}, J_B) \leq \varepsilon$, alors la solution réalisable $\bar{x}$ est ε-optimale,

sinon on remplace J_B par un nouveau support $\bar{J}_B$ de la manière suivante :

– Si $\theta^0 = \theta_{j_0}$, alors l'indice est devenu optimal. On pose donc $\bar{x} = x + \theta^0 l$, $\bar{J}_B = J_B$.

– Si $\theta^0 = \theta_{j_1}$, on pose :

$$\bar{x} = x + \theta^0 l \quad \text{et} \quad \bar{J}_B = (J_B \setminus j_1) \cup j_0.$$

2.6.5 Algorithme de résolution d'un problème de PLE par la méthode de support

L'objectif de cet algorithme est de construire une solution optimale pour le problème (2.8), en se basant sur le travail [18].

Le principe est le même que celui des algorithmes de plans sécants. On commence par résoudre le problème relaxé (2.9) en utilisant l'algorithme direct de support. Si la solution optimale du (PLR) est entière, alors elle est également une solution optimale du (PLE) ; donc la résolution est terminée. Sinon, si une ou plusieurs variables ne sont pas entières,

alors on génère une coupe valide. Cette contrainte supplémentaire est ajoutée au programme PLR et on résout de nouveau le problème augmenté. Si la solution obtenue est à valeurs entières, la résolution est terminée. Sinon, on doit encore générer une coupe et la procédure sera ainsi répétée jusqu'à l'obtention d'une solution optimale à composantes entières.

Soit $\{x, J_B\}$ un plan de support optimal du problème (2.9).
Suivant les valeurs des composantes du vecteur des estimations E_N, on définit les ensembles d'indices suivants :

$J_N^+ = \{j \in J_N : E_j > 0\}, \qquad J_N^- = \{j \in J_N : E_j < 0\},$
$J_{N0} = \{j \in J_N : E_j = 0\}, \qquad J_{N0} = J^+ \cup J^-,$
où $J^+ = \{j \in J_{N0} : x_j - d_j^- \leq d_j^+ - x_j\}$, et $J^- = \{j \in J_{N0} : x_j - d_j^- > d_j^+ - x_j\}$.

En fonction de J_B, on construit les fonctions pour tout $i \in J_B$:

$$Z_i(x) = -\left\{\tilde{b}_i - \sum_{j \in J_N^+ \cup J^+} x_{ij} d_j^- - \sum_{j \in J_N^- \cup J^-} x_{ij} d_j^+\right\} + \sum_{j \in J_N^+ \cup J^+} \{x_{ij}\}(x_j - d_j^-) + \sum_{j \in J_N^- \cup J^-} \{-x_{ij}\}(-x_j + d_j^+), \tag{2.23}$$

où :

$$\tilde{b} = (A(I, J_B))^{-1} b = A_B^{-1} b \text{ et } X = (x_{ij}, i \in J_B, j \in J_N) = (A(I, J_B))^{-1} A_N = A_B^{-1} A_N.$$

Proposition 2.6.2 *$Z_i(x^e)$ est un nombre entier, quelque soit $\{x^e, J_B\}$ plan de support du problème (2.8) et de plus, on a : $Z_i(x^e) \geq 0$, $\forall i \in J_B$.*

Preuve 2.6.3 *Soit x^e une solution réalisable du problème (2.8), c'est à dire :*

$$Ax^e = b, d^- \leq x^e \leq d^+ \text{ et } x_j^e \text{ entier } \forall\ j \in J.$$

On a :

$$\begin{aligned} Ax^e = b &\Rightarrow A_B x_B^e + A_N x_N^e = b, \\ &\Rightarrow x_B^e = A_B^{-1}(b - A_N x_N^e), \\ &\Rightarrow x_B^e = A_B^{-1} b - A_B^{-1} A_N x_N^e = \tilde{b} - X x_N^e, \\ &\Rightarrow x_i^e = \tilde{b}_i - \sum_{j \in J_N} x_{ij} x_j^e, \quad \forall i \in J_B, \end{aligned}$$

$$\Rightarrow \tilde{b}_i = x_i^e + \sum_{j \in J_N} x_{ij} x_j^e, \quad \forall i \in J_B. \tag{2.24}$$

En remplaçant $\tilde{b}$ dans la formule (2.23), on aura pour $i \in J_B$:

$$Z_i(x^e) = -\left\{ \tilde{b}_i - \sum_{j \in J_N^+ \cup J^+} x_{ij} d_j^- - \sum_{j \in J_N^- \cup J^-} x_{ij} d_j^+ \right\} + \sum_{j \in J_N^+ \cup J^+} \{x_{ij}\}(x_j^e - d_j^-) + \sum_{j \in J_N^- \cup J^-} \{-x_{ij}\}(-x_j^e + d_j^+),$$

$$= -\left\{ x_i^e + \sum_{j \in J_N} x_{ij} x_j^e - \sum_{j \in J_N^+ \cup J^+} x_{ij} d_j^- - \sum_{j \in J_N^- \cup J^-} x_{ij} d_j^+ \right\} + \sum_{j \in J_N^+ \cup J^+} \{x_{ij}\}(x_j^e - d_j^-)$$

$$+ \sum_{\substack{j \in J_N^- \cup J^- \\ \{x_{ij}\} \neq 0}} (1 - \{x_{ij}\})(-x_j^e + d_j^+),$$

car $\{-a\} = 1 - \{a\}$, $\forall a \in \mathbb{R} \setminus \mathbb{Z}$. Comme $\{a + k\} = \{a\}$, $\forall a \in \mathbb{R}$ et $\forall k \in \mathbb{Z}$, on aura :

$$Z_i(x^e) = -\left\{ \sum_{j \in J_N^+ \cup J^+} x_{ij}(x_j^e - d_j^-) + \sum_{j \in J_N^- \cup J^-} x_{ij}(x_j^e - d_j^+) \right\} + \sum_{j \in J_N^+ \cup J^+} \{x_{ij}\}(x_j^e - d_j^-) + \sum_{j \in J_N^- \cup J^-} \{x_{ij}\}(x_j^e - d_j^+)$$

$$+ \sum_{\substack{j \in J_N^- \cup J^- \\ \{x_{ij}\} \neq 0}} (d_j^+ - x_j^e). \tag{2.25}$$

On sait que :

$$\{ \sum_{j \in J_N^+ \cup J^+} x_{ij}(x_j^e - d_j^-) + \sum_{j \in J_N^- \cup J^-} x_{ij}(x_j^e - d_j^+) \} \leq \{ \sum_{j \in J_N^+ \cup J^+} x_{ij}(x_j^e - d_j^-) \} + \{ \sum_{j \in J_N^- \cup J^-} x_{ij}(x_j^e - d_j^+) \},$$

$$\leq \sum_{j \in J_N^+ \cup J^+} \{x_{ij}(x_j^e - d_j^-)\} + \sum_{j \in J_N^- \cup J^-} \{x_{ij}(x_j^e - d_j^+)\},$$

$$\leq \sum_{j \in J_N^+ \cup J^+} \{x_{ij}\}(x_j^e - d_j^-) + \sum_{j \in J_N^- \cup J^-} \{x_{ij}\}(x_j^e - d_j^+).$$

Comme $\sum_{\substack{j \in J_N^- \cup J^- \\ \{x_{ij}\} \neq 0}} (d_j^+ - x_j^e) \geq 0$, alors on a $Z_i(x^e) \geq 0, \forall i \in J_B$ et x^e solution réalisable entière de (2.8).

D'autre part, on peut écrire :

$$\{\sum_{j\in J_N^+\cup J^+} x_{ij}(x_j^e-d_j^-)+\sum_{j\in J_N^-\cup J^-} x_{ij}(x_j^e-d_j^+)\}=\{\sum_{j\in J_N^+\cup J^+} x_{ij}(x_j^e-d_j^-)\}+\{\sum_{j\in J_N^-\cup J^-} x_{ij}(x_j^e-d_j^+)\}$$

$$-[\{\sum_{j\in J_N^+\cup J^+} x_{ij}(x_j^e-d_j^-)\}+\{\sum_{j\in J_N^-\cup J^-} x_{ij}(x_j^e-d_j^+)\}]$$

$$=\sum_{j\in J_N^+\cup J^+}\{x_{ij}(x_j^e-d_j^-)\}+\sum_{j\in J_N^-\cup J^-}\{x_{ij}(x_j^e-d_j^+)\}-[\sum_{j\in J_N^+\cup J^+}\{x_{ij}(x_j^e-d_j^-)\}]-[\sum_{j\in J_N^-\cup J^-}\{x_{ij}(x_j^e-d_j^+)\}]$$

$$-[\{\sum_{j\in J_N^+\cup J^+} x_{ij}(x_j^e-d_j^-)\}+\{\sum_{j\in J_N^-\cup J^-} x_{ij}(x_j^e-d_j^+)\}],$$

$$=\sum_{j\in J_N^+\cup J^+}\{x_{ij}\}(x_j^e-d_j^-)+\sum_{j\in J_N^-\cup J^-}\{x_{ij}\}(x_j^e-d_j^+)-\sum_{j\in J_N^+\cup J^+}[\{x_{ij}\}(x_j^e-d_j^-)]-\sum_{j\in J_N^-\cup J^-}[\{x_{ij}\}(x_j^e-d_j^+)]$$

$$-[\sum_{j\in J_N^+\cup J^+}\{x_{ij}(x_j^e-d_j^-)\}]-[\sum_{j\in J_N^-\cup J^-}\{x_{ij}(x_j^e-d_j^+)\}]-[\{\sum_{j\in J_N^+\cup J^+} x_{ij}(x_j^e-d_j^-)\}+\{\sum_{j\in J_N^-\cup J^-} x_{ij}(x_j^e-d_j^+)\}]$$

En remplaçant la formule (2.25), on aura :

$$Z_i(x^e)=\sum_{\substack{j\in J_N^-\cup J^-\\ \{x_{ij}\}\neq 0}}(d_j^+-x_j^e)+\sum_{j\in J_N^+\cup J^+}[\{x_{ij}\}(x_j^e-d_j^-)]+\sum_{j\in J_N^-\cup J^-}[\{x_{ij}\}(x_j^e-d_j^+)]-[\sum_{j\in J_N^+\cup J^+}\{x_{ij}(x_j^e-d_j^-)\}]$$

$$-[\sum_{j\in J_N^-\cup J^-}\{x_{ij}(x_j^e-d_j^+)\}]-[\{\sum_{j\in J_N^+\cup J^+} x_{ij}(x_j^e-d_j^-)\}+\{\sum_{j\in J_N^-\cup J^-} x_{ij}(x_j^e-d_j^+)\}] \quad \in \mathbb{Z}_+$$

Proposition 2.6.3 *Soit $\{x^0, J_B^0\}$ un plan de support optimal de (2.9). S'il existe un indice $i \in J_B^0$ tel que $Z_{i_1}(x^0) < 0$, alors l'inégalité*

$$Z_{i_1}(x) \geq 0,$$

donne une coupe valide.

Preuve 2.6.4 *Évident, car on a démontré que pour une solution réalisable x^e du problème (2.8), on a $Z_i(x^e) \geq 0$, $\forall i \in J_B$.*

Proposition 2.6.4 *Soit $\{x^0, J_B^0\}$ un plan de support optimal du problème (2.9). Si x^0 est une solution réalisable non entière tel que $\forall i \in J_B^0$, on a $Z_i(x^0) \geq 0$, alors x^0 est forcément une solution réalisable non basique.*

Preuve 2.6.5 *Si* $\{x^0, J_B^0\}$ *est un plan de support optimal basique du problème (2.9), alors dans ce cas on aura :*

$$x_j^0 = \begin{cases} d_j^-, & j \in J_N^+ \cup J^+, \\ d_j^+, & j \in J_N^- \cup J^-. \end{cases}$$

Soit un indice i_1 *tel que la composante* $x_{i_1}^0$ *est non entière, alors,* $0 < \{x_{i_1}^0\} < 1$. *L'expression de* $Z_{i_1}(x^0)$ *s'écrit comme suit :*

$$Z_{i_1}(x^0) = -\left\{\tilde{b}_{i_1} - \sum_{j \in J_N^+ \cup J^+} x_{i_1 j} d_j^- - \sum_{j \in J_N^- \cup J^-} x_{i_1 j} d_j^+\right\} + \sum_{j \in J_N^+ \cup J^+} \{x_{i_1 j}\}(d_j^- - d_j^-) + \sum_{j \in J_N^- \cup J^-} \{-x_{i_1 j}\}(-d_j^+ + d_j^+)$$

$$= -\left\{\tilde{b}_{i_1} - \sum_{j \in J_N^+ \cup J^+} x_{i_1 j} d_j^- - \sum_{j \in J_N^- \cup J^-} x_{i_1 j} d_j^+\right\}$$

D'autre part, on a :

$$\begin{aligned} & Ax^0 = b \\ \Rightarrow\ & A_B x_B^0 + A_N x_N^0 = b, \\ \Rightarrow\ & x_B^0 = A_B^{-1} b - A_B^{-1} A_N x_N^0, \\ & = \tilde{b} - X x_N^0, \\ \Rightarrow\ & \tilde{b} = x_B^0 + X x_N^0, \end{aligned}$$

$\Rightarrow \tilde{b}_i = x_i^0 + \sum_{j \in J_N^+ \cup J^+} x_{ij} d_j^- + \sum_{j \in J_N^- \cup J^-} x_{ij} d_j^+, \quad i \in J_B^0.$

D'où

$$Z_{i_1}(x^0) = -\{x_{i_1}^0\} < 0, \text{ d'où la contradiction.}$$

Proposition 2.6.5 *Quand la solution réalisable* x^0 *est basique, donc on a :* $Z_i(x^0) \leq 0, \ \forall i \in J_B$.

Schéma de l'algorithme

ENTRÉE : Une instance d'un problème de PLE ;
SORTIE : Une solution optimale x^0.

Étape 1 poser $t = 0, I_0 = I = \{1, 2, ..., m\}$ et $J^0 = J = \{1, 2, ..., n\}$.
Soit $\{x^*, J_B^*\}$ un plan de support initial du problème (2.9).
Donc on construit un plan de support $\{x^0, J_B^0\}$ avec la méthode directe de support.
Si $x^0 \in \mathbb{Z}^n$, arrêter l'algorithme : x^0 est une solution optimale du problème (2.8).

Sinon, aller à l'étape 2.

Étape 2 On construit les fonctions $Z_i(x^0)$, pour tout $i \in J_B^0$.

Étape 3 Deux cas peuvent se présenter :
(i) $\exists i_1 \in J_B^0 :$ $Z_{i_1}(x^0) < 0$, dans ce cas l'inégalité :

$$Z_{i_1}(x) \geq 0 \tag{2.26}$$

donne une coupe valide.
On incrémente t, $t = t + 1$.
On ajoute la contrainte (2.26) au système (2.9) et on résout le problème :

$$\begin{cases} max \;\; z(x) = c'x, \\ \qquad\quad Ax = b, \\ \qquad\quad Z_{i_1}(x) \geq 0, \\ \qquad\quad d^- \leq x \leq d^+. \end{cases} \tag{2.27}$$

On commence par le plan de support $\{\bar{x}, \bar{J}_B\}$, avec

$$\bar{x} = x^* + \lambda(x^0 - x^*), \quad 0 < \lambda < 1,$$

où λ sera trouvé à partir de l'équation $Z_{i_1}(\bar{x}) = 0$.
– Comme on a ajouté une inégalité au système (2.27), on rajoute une variable d'écart x_{n+t}. On prend alors le support $\bar{J}_B = J_B^0 \cup \{n + t\}$.
On pose $I_t = I_{t-1} \cup \{m + t\}$ et $J^t = J^{t-1} \cup \{n + t\}$.
Soit $\{x^1, J_B^1\}$ le plan de support optimal de (2.27).
On refait alors le processus de résolution avec $\{x^1, J_B^1\}$ le nouveau plan de support.
(ii) $Z_i(x^0) \geq 0, \quad \forall i \in J_B^0$.
Dans ce cas,on distingue deux possibilités :
– x^0 est non entier et basique, cette situation ne peut pas avoir lieu d'après la proposition (2.6.4). C'est à dire, dans le cas de solutions basiques, le deuxième cas ne se présente pas.
– x^0 est non entier et non basique (la solution optimale x^0 n'est pas unique).
Dans ce cas, il faut construire une solution x^{0B} basique, et construire par cette dernière une coupe régulière comme dans le cas (i).

Montrons que $\{\bar{x}, \bar{J}_B\}$ est bien un plan de support pour le problème (2.27).

Preuve 2.6.6 *on a*

$$A\bar{x} = A(x^* + \lambda(x^0 - x^*)) = Ax^* + \lambda A(x^0 - x^*) = b + \lambda A\Delta x = b,$$

où $A\Delta x = 0$*, vu que* x^0 *et* x^* *sont deux solutions réalisables de (2.9).*
Le vecteur $\bar{x}$ *vérifie la contrainte principale* $A\bar{x} = b$*. Il vérifie aussi la contrainte* $Z_{i_1}(\bar{x}) \geq 0$*, puisque par construction, le pas* λ *est tel que* $Z_{i_1}(\bar{x}) = 0$*.*
Pour que $\bar{x}$ *soit une solution réalisable du problème (2.27), il doit en plus appartenir à* $S = \{x \in \mathbb{R}^n : d^- \leq x \leq d^+\}$*. En effet, comme* S *est convexe et* x^* *et* $x^0 \in S$*, alors*

$$\bar{x} = \lambda x^0 + (1 - \lambda)x^* \in S.$$

D'où $\bar{x}$ *est une solution réalisable du problème (2.27).*
$\bar{J}_B$ *est bien un support, puisque :*

$$detA(I_t, \bar{J}_B) = detA(I_{t-1} \cup \{m + t\}, J_B^0 \cup \{n + t\}) = detA(I_{t-1}, J_B^0) \neq 0.$$

Calcul de λ à partir de l'équation $Z_{i_1}(\bar{x}) = 0$

$$-\{\tilde{b}_{i_1} - \sum_{j \in J_N^- \cup J^-} x_{i_1 j} d_j^+ - \sum_{j \in J_N^+ \cup J^+} x_{i_1 j} d_j^-\} + \sum_{j \in J_N^+ \cup J^+} \{x_{i_1 j}\}(x_j^* + \lambda(x_j^0 - x_j^*) - d_j^-)$$
$$\sum_{j \in J_N^- \cup J^-} \{-x_{i_1 j}\}(-x_j^* - \lambda(x_j^0 - x_j^*) + d_j^+) = 0,$$
$$Z_{i_1}(x^*) + \lambda \sum_{j \in J_N^+ \cup J^+} \{x_{i_1 j}\}(x_j^0 - x_j^*) - \lambda \sum_{j \in J_N^- \cup J^-} \{-x_{i_1 j}\}(x_j^0 - x_j^*) = 0,$$
$$Z_{i_1}(x^*) + \lambda(\sum_{j \in J_N^+ \cup J^+} \{x_{i_1 j}\}(x_j^0 - x_j^*) + \sum_{j \in J_N^- \cup J^-} \{x_{i_1 j}\}(x_j^0 - x_j^*) - \sum_{\substack{j \in J_N^- \cup J^- \\ \{x_{i_1 j}\} \neq 0}} (x_j^0 - x_j^*)) = 0,$$
$$Z_{i_1}(x^*) + \lambda(\sum_{j \in J_N} \{x_{i_1 j}\}(x_j^0 - x_j^*) - \sum_{\substack{j \in J_N^- \cup J^- \\ \{x_{i_1 j}\} \neq 0}} (x_j^0 - x_j^*)) = 0,$$

d'où

$$\lambda = \frac{Z_{i_1}(x^*)}{\sum\limits_{\substack{j \in J_N^- \cup J^- \\ \{x_{i_1 j}\} \neq 0}} (x_j^0 - x_j^*) - \sum\limits_{j \in J_N} \{x_{i_1 j}\}(x_j^0 - x_j^*)}. \tag{2.28}$$

D'autre part,on a :

$$Z_{i_1}(x^0)-Z_{i_1}(x^*) = \sum_{j\in J_N^+\cup J^+}\{x_{i_1j}\}(x_j^0-d^-)+\sum_{j\in J_N^-\cup J^-}\{-x_{i_1j}\}(-x_j^0+d^+)-\sum_{j\in J_N^+\cup J^+}\{x_{i_1j}\}(x_j^*-d^-)$$

$$\begin{aligned}&-\sum_{j\in J_N^-\cup J^-}\{-x_{i_1j}\}(-x_j^*+d^+),\\ &=\sum_{j\in J_N^+\cup J^+}\{x_{i_1j}\}(x_j^0-x^*)-\sum_{j\in J_N^-\cup J^-}\{-x_{i_1j}\}(x_j^0-x^*),\\ &=\sum_{j\in J_N^+\cup J^+}\{x_{i_1j}\}(x_j^0-x_j^*)-\sum_{j\in J_N^-\cup J^-}\{-x_{i_1j}\}(x_j^0-x_j^*).\end{aligned}$$

La formule (2.28) est équivalente à

$$\lambda=\frac{Z_{i_1}(x^*)}{Z_{i_1}(x^*)-Z_{i_1}(x^0)}.$$

Exemple 2.6.4

Soit à résoudre le problème de PLE suivant par la méthode de support :

$$\begin{cases} max\ z(x)=3x_1+x_2+2x_3+3x_4,\\ -x_1+3x_2+x_3-2x_4+x_5=17,\\ 7x_1+\qquad+3x_3+x_4+x_6=23,\\ 0\le x_i\le 12,\quad x_i \text{ entier},\quad i=\overline{1,6}.\end{cases} \tag{2.29}$$

Le problème relaxé associé est :

$$\begin{cases} max\ z(x)=3x_1+x_2+2x_3+3x_4,\\ -x_1+3x_2+x_3-2x_4+x_5=17,\\ 7x_1+\qquad+3x_3+x_4+x_6=23,\\ 0\le x_i\le 12,\quad i=\overline{1,6}.\end{cases} \tag{2.30}$$

Étape 1 : On résout par la méthode directe de support le problème (2.30).

Soit $x^*=(0,0,5,0,12,8)'$ et $J_B^*=\{3,1\}$ un plan initial de support du problème (2.30) avec $z(x^*)=10$.

Après 5 itérations, nous avons trouvé le plan de support optimal suivant :

$$x^0=(0,12,11/3,12,4/3,0)'\ \ J_B^0=\{3,5\}$$

avec $z(x^0) = 166/3$.

Étape 2 : Le plan x^0 est non entier, on rajoute alors au programme (2.30) une coupe.

Itération 1 Le vecteur des estimations vaut :

$$E_N^0 = (E_1^0, E_2^0, E_4^0, E_6^0)' = (5/3, -1, -7/3, 2/3)', J_N^0 = \{1, 2, 4, 6\}.$$

On partitionne l'ensemble J_N^0 comme suit :

$J_N^+ = \{1, 6\}$,

$J_N^- = \{2, 4\}$,

$J^+ = J^- = \emptyset$.

On calcule $\tilde{b}$: $\tilde{b} = \begin{pmatrix} 23/3 \\ 28/3 \end{pmatrix}$.

On calcule X : $X = \begin{pmatrix} 7/3 & 0 & 1/3 & 1/3 \\ -10/3 & 3 & -7/3 & -1/3 \end{pmatrix}$.

On calcule alors le vecteur des fonctions $z_i(x^0)$, $i \in J_B^0$:

on trouve

$$Z_B(x^0) = -\left\{ \begin{pmatrix} 23/3 \\ 28/3 \end{pmatrix} - \begin{pmatrix} 0 & 1/3 \\ 3 & -7/3 \end{pmatrix} \begin{pmatrix} 12 \\ 12 \end{pmatrix} \right\} = (-2/3, -1/3)'.$$

On se trouve dans le cas 1. Alors on rajoute aux contraintes (2.30) la contrainte $Z_{i1}(x) \geq 0$ tel que i_1 est l'indice qui correspond au $\min Z_i(x^0)$, $i \in J_B$.

$$Z_1(x) = -2/3 + (1/3, 1/3) \begin{pmatrix} x_1 \\ x_6 \end{pmatrix} + (0, 2/3) \begin{pmatrix} -x_2 + 12 \\ -x_4 + 12 \end{pmatrix} \geq 0.$$

D'où la contrainte suivante : $x_1 - 2x_4 + x_6 - 3x_7 = -22$.

Le programme résultant est :

$$\begin{cases} max\ z(x) = 3x_1 + x_2 + 2x_3 + 3x_4, \\ -x_1 + 3x_2 + x_3 - 2x_4 + x_5 = 17, \\ 7x_1 + \qquad +3x_3 + x_4 + x_6 = 23, \\ x_1 - 2x_4 + x_6 - 3x_7 = -22, \\ 0 \leq x_i \leq 12, \quad i = \overline{1,6}, \ \ x_7 \geq 0. \end{cases} \tag{2.31}$$

On résout le programme (2.31) en commençant par une solution notée $\bar{x}$ calculée comme suit :$\bar{x} = x^* + \lambda(x^o - x^*)$, où $\lambda = \frac{15}{16}$,
donc

$$\bar{x} = (0, \frac{45}{4}, \frac{15}{4}, \frac{45}{4}, 2, \frac{1}{2})'.$$

Après l'ajout de la variable d'écart x_7,

$$\bar{x} = (0, \frac{45}{4}, \frac{15}{4}, \frac{45}{4}, 2, \frac{1}{2}, 0)',$$

et $\bar{J_B} = \{3, 5, 7\}$.
Après 6 itérations, nous avons obtenu une solution optimale de support qui est :

$$x^0 = (0, 12, 3, 12, 2, 2, 0)', \quad z(x^0) = 54.$$

Cette solution est entière, alors le processus s'arrête.

2.7 Conclusion

Nous avons présenté dans ce chapitre les différentes notions élémentaires de la programmation linéaire en nombres entiers. Nous avons abordé également les méthodes de résolution des problèmes de PLE. Nous avons donné les schémas généraux des algorithmes exacts pour la résolution de ces problèmes.

Chapitre 3

Méthodes approchées pour la résolution des problèmes de PLE

3.1 Introduction

Les problèmes de PLE appartiennent à la classe des problèmes NP-difficiles. Cela nécessite de grands efforts de calcul pour trouver une solution optimale pour des problèmes de taille importante [24].

Très souvent, il est plus important de trouver une solution acceptable, au lieu d'attendre longtemps pour obtenir la solution optimale. Dans la pratique, certaines contraintes sont flexibles et les algorithmes exactes peuvent résoudre le problème mais cela peut prendre beaucoup de temps et peut être coûteux pour des applications réelles. En revanche, Les algorithmes approchés ne sont pas si sensibles aux petits changements et s'adaptent facilement avec cette flexibilité.

Les algorithmes approchés trouvent un large champ d'applications. Ils pourraient être utilisés pour trouver une solution initiale appropriée ou pour rétrécir le domaine des solutions réalisables et se diriger vers la recherche d'une solution optimale. Un grand nombre d'algorithmes approchés a été créé pour la résolution de gros problèmes de PLE dans la pratique, sans aucune garantie d'optimalité de la solution finale [25].

Dans ce chapitre, nous parlerons d'un type spécifique d'algorithmes approchés. Il s'agit d'approches heuristiques. Nous donnerons un aperçu sur les heuristiques en générale, puis passer aux différentes heuristiques utilisées dans la résolution des problèmes de PLE.

3.2 Un aperçu sur les heuristiques

De nombreux chercheurs comme Ackoff [44], Müller-Merbach[45] et autres ont souligné que l'utilisation de modèles mathématiques pour faciliter la prise de décision par rapport à des situations réelles impliquait réellement différentes étapes comme nous le voyons dans cette figure :

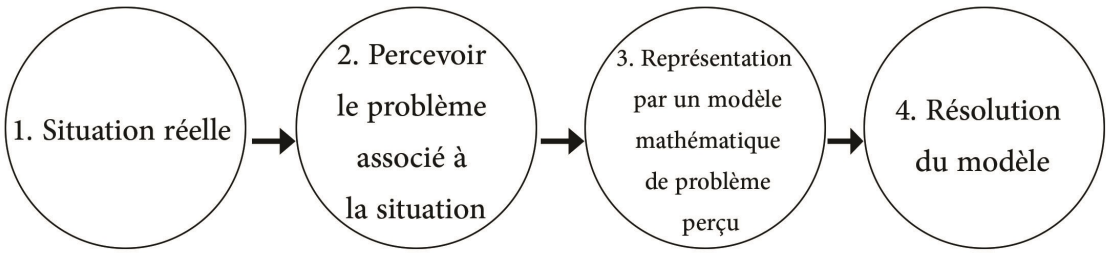

FIGURE 3.1 – D'une situation réelle à la solution du problème

3.2.1 Pourquoi utiliser une heuristique ?

En revenant à la figure ci-dessus, l'intérêt de l'étape 3 devrait être de construire un modèle mathématique, réellement représentatif du problème perçu, qui soit aussi simple que possible. Bien que la simplicité soit souhaitée, pour être adapté à la situation perçue, le modèle conçu peut être d'une forme difficile telle qu'il est impossible de trouver sa solution optimale. Dans une telle situation, pour parvenir à l'optimalité, on est obligé de simplifier le modèle en introduisant des hypothèses. De plus, il y a la question liée à l'inexactitude des données nécessaires associées. Connaissant ces facteurs, il est sans doute préférable d'obtenir une solution raisonnable à un modèle plus précis que de chercher la solution optimale à un modèle incorrect ou trop simplifié du problème du monde réel.

Ainsi, les heuristiques permettent d'utiliser des modèles plus représentatifs des problèmes rencontrés en réalité. Voici quelques raisons d'utiliser des heuristiques pour résoudre un problème [25] :

(i) Simples à mettre en œuvre : Woolsey et Swanson [26] disent que les gens préfèrent de vivre avec un problème qu'ils ne peuvent pas résoudre que d'accepter une solution qu'ils ne peuvent pas comprendre. Utiliser des règles de décision qui sont susceptibles de faciliter la compréhension, mais cela ne signifie pas que les heuristiques doivent nécessairement être simples de nature. Cependant, pour des problèmes très complexes, elles sont peut être pas utiles.

(ii) Améliorent les résultats : Il faut être vigilant que la solution trouvée à l'aide d'une heuristique soit meilleure que les résultats qui existaient déjà.

(iii) Résultats rapides : Les heuristiques peuvent être plus rapides que d'autres algorithmes d'optimisation.

(iv) Robustesse : Les heuristiques peuvent être moins sensibles aux variations des caractéristiques du problème et de la qualité des données.

(v) S'utilisent avec d'autres méthodes d'optimisation : On peut combiner facilement une heuristique avec une méthode exacte, on l'appelle dans ce cas une « méthode hybride ». Elles peuvent être utilisées avec ces méthodes de deux manières. Premièrement, elles peuvent fournir une bonne solution initiale. Deuxièmement, elles peuvent fournir des bornes.

3.2.2 Types basiques des heuristiques

Il y a sept types basiques des heuristiques qui sont :

- Des solutions générées aléatoirement ;
- Décomposition ou partitionnement ;
- Méthodes inductives ;
- Méthodes pour réduire l'espace des solutions ;
- Méthodes approximatives ;
- Méthodes constructives ;
- Méthodes de recherche locale.

En effet, il est souvent logique de mélanger plus d'un type d'heuristique dans la résolution d'une classe spécifique de problèmes. En outre, il peut être utile d'utiliser deux ou plusieurs méthodes distinctes en parallèle pour résoudre le même problème, en choisissant la meilleure des solutions.

Le choix de telles approches heuristiques dépend de plusieurs facteurs, citons :

- Le domaine de décision est stratégique, tactique ou opérationnel ;
- La fréquence avec laquelle la décision est prise ;
- Le temps de développement disponible ;
- Les qualifications analytiques du décideur impliquées ;
- La taille du problème ;
- L'absence ou présence d'éléments stochastiques.

3.2.3 Comment évaluer une heuristique ?

Une bonne heuristique est bien souvent spécifique à un problème : elle est par exemple la traduction du savoir d'un spécialiste du domaine. Apprendre à construire de telles heuristiques se fait par l'expérience de nombreux cas concrets.

Principalement, les heuristiques sont conçues en validant étape par étape leur proximité avec la réalité du problème qu'elles résolvent. Il est intéressant de valider la justesse de son heuristique avec des solutions exactes (quand on les connaît) ou avec des solutions fournies par l'expérience. Néanmoins, il est théoriquement impossible d'évaluer réellement une solution heuristique [27].

3.3 Heuristiques en PLE

Dans la PLE, une heuristique est un algorithme approximatif conçu pour trouver rapidement de bonnes solutions, sans qu'elles soient optimales. Le mot "heuristique" invoque le concept de recherche ciblée, parce que le mot dérive du grec "heuriskein" qui signifie "découvrir". Shapiro (1968) [38] fait la distinction entre les heuristiques spécifiques aux problèmes qui utilisent des « règles empiriques »pour arriver à de bonnes solutions réalisables aux problèmes de PLE et les méthodes à usage général pour la recherche d'un espace des solutions réalisables connues sous le nom d'algorithme génétique. Il suggère que ce dernier "peut être combiné avec des heuristiques spécifiques aux problèmes pour améliorer leur efficacité" [10].
Quand on utilise une heuristique pour pouvoir contrôler une autre, l'algorithme obtenu est connu sous le nom de *métaheuristique*.

3.3.1 Heuristiques primales

Les heuristiques primales sont des techniques importantes pour la résolution des problèmes de PLE. Elles consistent à générer de "bonnes" solutions réalisables en utilisant les outils de l'optimisation. Elles sont couramment utilisées par les solveurs commerciaux, ces dernières années, et la capacité de ces solveurs à trouver rapidement dans l'arborescence de recherche des solutions de très haute qualité, s'est considérablement améliorée [28].
Nous distinguons parmi les heuristiques primales qui existent les types suivants :

Les heuristiques d'arrondi. À partir d'une solution réalisable pour le problème relaxé, on arrondit supérieurement ou inférieurement les valeurs fractionnaires des variables dans la but de produire une solution entière satisfaisant toujours les contraintes linéaires. Ceci peut être fait d'une manière simple et très rapide ou d'une manière plus complexe,

heuristique "Diving". Comparativement à une heuristique d'arrondi, "the diving" est effectuée d'une manière itérative : après avoir arrondi une ou plusieurs variables, la relaxation linéaire est à nouveau résolue et le processus sera réitéré. Dans cette catégorie, Achterberg [29]distingue l'arrondi dur dans lequel les variables sont réellement fixées à une valeur spécifiée et l'arrondissement léger dans lequel l'effet est obtenu implicitement en changeant de manière appropriée les coefficients de la fonction objectif des variables à arrondir. Un algorithme relevant de cette dernière catégorie est l'heuristique de faisabilité proposée pour les problèmes en 0-1, puis étendue aux programmes en nombres entiers et mixtes généraux. L'idée de l'algorithme est de satisfaire alternativement les contraintes linéaires (en résolvant les relaxations linéaires) ou les contraintes d'intégrité (en appliquant l'arrondi). Les relaxations sont obtenues en remplaçant la fonction objectif d'origine par une prenant en compte la distance par rapport à la solution entière arrondie actuelle. Si la trajectoire des solutions arrondies intersecte l'une des voisinages, alors une solution réalisable par rapport aux contraintes linéaires et contraintes d'intégrité a été trouvée.

les heuristiques d'amélioration. Les heuristiques de cette catégorie sont conçues pour améliorer la qualité de la solution actuelle, c'est-à-dire qu'elles exploitent cette solution ou un groupe de solutions de manière à obtenir une ou des solutions meilleures. Ces heuristiques sont généralement des méthodes de recherche locale. Les heuristiques d'amélioration ont des relations avec les techniques dites métaheuristiques qui se sont avérées très efficaces sur les problèmes combinatoires difficiles.

3.3.2 Heuristiques basées sur des relaxations

Différentes heuristiques basées sur des relaxations permettent d'obtenir de bonnes solutions pour les problèmes de PLE. Une heuristique simple consistant à arrondir une solution optimale de la relaxation en continu. L'idée derrière cette approche est qu'une solution optimale de la relaxation en continu peut être proche d'une solution optimale du

problème.

Deux types d'approche peuvent être envisagés. Dans le premier cas, la solution de la relaxation en continu est *tronquée* (prendre juste la partie entière $[x_j]$) de manière à générer une solution réalisable du problème. Dans le second cas, la solution est *arrondie* (prendre : $[x_j]$ si $0 \leq \{x_j\} \leq 1/2$ et $[x_j]+1$ sinon), ce qui amène dans certains cas à une solution non réalisable. Il est également possible de choisir une méthode d'arrondi, et ensuite d'appliquer un opérateur adapté au problème considéré pour transformer la solution obtenue en solution réalisable si elle ne l'est pas.

Ce type d'heuristiques peut être utilisé comme solution initiale dans un algorithme plus complet. Un avantage de ces heuristiques est qu'elles permettent d'obtenir une solution réalisable très rapidement.

Exemple 3.3.1 *[35]*

$$\begin{cases} max\ z(x) = 20x_1 + 18x_2 + 15x_3 + 14x_4 + 12x_5 + 9x_6 + 7x_7 + 5x_8 + 3x_9 + 2x_{10}, \\ s.c. \quad 15x_1 + 16x_2 + 12x_3 + 12x_4 + 10x_5 + 10x_6 + 8x_7 + 5x_8 + 4x_9 + 3x_{10} \leq 45, \\ \qquad 22x_1 + 21x_2 + 16x_3 + 14x_4 + 15x_5 + 7x_6 + 5x_7 + 2x_8 + 4x_9 + 4x_{10} \leq 50, \\ \qquad 18x_1 + 20x_2 + 15x_3 + 10x_4 + 9x_5 + 8x_6 + 2x_7 + 6x_8 + 2x_9 + 5x_{10} \leq 40, \\ \qquad x_j \in \{0,1\} \quad j = 1, ..., 10. \end{cases}$$

La valeur optimale de ce problème vaut 50 avec $x^* = (1,0,0,1,0,1,1,0,0,0)'$. Une solution optimale de la relaxation en continu est donnée par : $\tilde{x} = (0.90, 0, 0, 1, 0.58, 0.07, 1, 1, 0, 0)'$ de valeur $z(\tilde{x}) = 51.60$. Si nous appliquons une heuristique d'arrondi à cette solution, nous obtenons la solution x^1 suivante : $x^1 = (1,0,0,1,1,0,1,1,0,0)'$, de valeur $z(x^1) = 58$ qui n'est pas réalisable. Si nous appliquons cette fois une troncature de la solution $\tilde{x}$, nous obtenons la solution $x^2 = (0,0,0,1,0,0,1,1,0,0)'$, de valeur $z(x^2) = 26$, réalisable mais très éloignée de la valeur optimale.

3.3.3 Approches de recherche locale

Les heuristiques de recherche locale commencent par une solution réalisable donnée et par des transformations élémentaires dans une ou quelques coordonnées, tentent d'améliorer la valeur de la fonction objectif. Par conséquent, ce type d'heuristique applique une

règle pour sélectionner un élément d'un ensemble. Woolsey (1972) a observé que beaucoup de ceux qui résolvent réellement les problèmes de la programmation entière se retournent vers des méthodes heuristiques pour obtenir de bonnes solutions de démarrage, suivies après par l'une des méthodes de branchement (*Branch-and-Bound*, *Branch-and-Cut*,etc.). Généralement, l'heuristique de recherche locale a trois rôles dans la PLE :

1. Localiser une solution réalisable comme point de départ pour un algorithme de PLE.
2. Les méthodes de recherche locale elles-mêmes peuvent bénéficier d'une solution de départ réalisable.
3. Les méthodes de recherche locale peuvent être combinées avec d'autres méthodes de recherche telles que les algorithmes génétiques ou les méthodes exactes, comme *Branch-and-Cut*.

3.3.3.1 Le voisinage d'une solution

Définition 3.3.1 *Le voisinage d'une solution donnée x est l'ensemble de solutions réalisables du problème de départ qui sont obtenues à partir de x via une transformation élémentaire, noté $N(x)$.*

Observation 3.3.1 *On dit que ces solutions (les voisinages) sont* proches *de x. La transformation élémentaire nécessite d'être définie selon les besoins de l'algorithme. La taille d'un voisinage est variable et est au maximum égal au cardinal de l'ensemble des solutions réalisables.*

Des exemples de modèles de la programmation entière et mixtes pour lesquels les algorithmes de recherche locale ont été développés sont comme suit :

- Ordonnancement en temps réel des travaux sur des modèles mixtes de chaînes d'assemblage (Bolat et al., 1994) [30] ;
- Problème de "trim-loss" dans les rouleaux de papier (Ramirez-Beltran et Aquilar-Ruggiero, 1997) [31] ;
- Ordonnancement de capacité de production (Walser et al., 1998) [32] ;
- Ordonnancement de type jobshop avec des coûts retardés et avancés (Danna et al,. 2003) [33] ;
- Problème de tournée de véhicules avec fenêtre de temps (Danna, 2004) [34].

3.3.4 Métaheuristiques

L'intelligence artificielle est un domaine d'étude vaste et important. La discussion ici est limitée à trois heuristiques sans détails qui se sont avérées d'une grande valeur pour résoudre les problèmes de PLE ainsi que les problèmes combinatoires :

- Recherche tabou [46] ;
- Recuit simulé [39] ;
- Algorithmes génétiques [11].

3.3.5 Heuristique basée sur la génération de colonnes

Avant de parler sur cette heuristique, il faut d'abord donner un aperçu sur la méthode dite « génération de colonnes ».

3.3.5.1 La génération de colonnes [36] [37]

La génération de colonnes, ou programmation linéaire généralisée est une méthode itérative utilisée pour résoudre des problèmes contenant un nombre important de variables. L'idée centrale de cette technique vient du fait que lors de la résolution d'un programme linéaire par la méthode de simplexe, plusieurs variables sont hors base et sont nulles à l'optimum, et peuvent donc être négligées.
Reposant sur cette particularité, la génération de colonnes consiste à ne manipuler, à la fois, qu'un sous ensemble de variables (colonnes) de petite taille et à identifier les variables entrant en base au cours de la résolution sans les énumérer explicitement. En effet, le mécanisme de génération de colonnes s'effectue au sein d'un algorithme, appelé algorithme générateur ou aussi "algorithme de pricing" qui résout généralement un ensemble de sous problèmes indépendants afin de filtrer les meilleures colonnes hors formulation actuelle du programme linéaire, c'est-à-dire celles qui sont susceptibles d'améliorer la solution courante. Ces colonnes correspondent aux variables dont les coûts réduits sont positifs (problème de maximisation). Autrement dit, la génération de colonnes est basée sur la décomposition du problème original en un problème maître restreint (PMR) et un sous-problème. Le PMR contient seulement un sous-ensemble de variables et le sous-problème est résolu pour déterminer si des colonnes peuvent être rajoutées au PMR ou alors affirmer le contraire.

Considérons le PL, écrit sous forme standard comme suit :

$$(PL)\begin{cases}\max z = cx,\\ s.c. \qquad Ax = b,\\ \qquad\qquad x \geq 0.\end{cases}$$

On suppose que $n >> m$ et la matrice A ne peut être représentée explicitement mais que l'on dispose d'un outil pour caractériser ses colonnes. **N** est l'ensemble de variables de (PL).

Considérons ensuite un sous-ensemble Ω des variables de (PL) et notons par A' la sous-matrice de A correspondant à Ω et par E_j le coût réduit d'une variable hors base x_j, défini par :

$$E_j = u'a_j - c_j,$$

où u est le vecteur multiplicateur du simplexe (variables duales) associés à la base optimale du problème restreint courant.

La méthode de génération de colonnes est décrite dans l'algorithme suivant :

ENTRÉE : Une instance du problème (PL) ;

SORTIE : Une solution optimale de (PL).

---

1. ***Initiation*** : B_0 := base réalisable initiale ; k=0 ; pricing = vrai.
2. ***Tant que*** (pricing=vrai) ***faire***
   (a) Calculer la solution de base $\overline{x} = B_k^{-1}b$;
   (b) Calculer les variables duales $u = c_{B_k}B_k^{-1}$;
   (c) Résolution du problème "pricing" :
   Pour $(j \in N \setminus \Omega)$
   i. **Si** $E_j = u'a_j - c_j \geq 0$ **alors**
   pricing = faux ;
   ii. **Sinon**
   Ajouter la variable j avec $E_j < 0$ à Ω ;
   Ajouter la colonne a_j à A' ;
   pricing = vrai ;
   (d) $k = k + 1$.
   Fin Tant que
3. Sortir avec une solution optimale de (PL).

Notons que l'efficacité de la méthode de génération de colonnes est très dépendante de l'algorithme générateur. En effet, ce dernier revient souvent à résoudre des sous-problèmes, qui sont NP-difficiles. Une résolution approchée peut être envisagée dans ce cas.

3.3.5.2 L'heuristique créée à l'aide de génération de colonnes

La génération de colonnes appliquée à un problème maître en nombres entiers fournit une solution qui n'est pas toujours entière. Pour obtenir une solution entière, une approche heuristique a été mise au point, consistant à effectuer une recherche arborescente par séparation et évaluation sur le PMR obtenu après le processus de génération de colonnes. On est donc confronté au choix de politique de séparation. Cependant, la solution ainsi trouvée peut ne pas être optimale ni même réalisable pour le problème initial. En effet, des colonnes, hors la formulation courante, sont susceptibles d'améliorer la solution (ou de rendre le problème faisable) au cours de la recherche arborescente.

3.3.6 Algorithmes gloutons

Un algorithme glouton est un algorithme constructif qui cherche à construire une solution pas à pas, en utilisant les données du problème. Après chaque prise de décision, on résout alors le sous-problème qui en découle :

- sans jamais revenir sur ses décisions ;
- en prenant à chaque étape la solution qui semble la meilleure localement ;
- en espérant obtenir une solution optimale.

Dans certains cas, la solution obtenue n'est pas optimale, on parle plutôt d'heuristique gloutonne, et dans le cas où elle l'est, d'algorithme glouton exact.

3.3.6.1 Le principe général d'une méthode gloutonne

On est souvent dans le cadre de problèmes d'optimisation. Plus précisément, on est le plus souvent dans le cas suivant :

- On a un ensemble fini d'éléments, E.
- Une solution à notre problème est construite à partir des éléments de E : c'est par exemple une partie de E ou un multi-ensemble d'éléments de E ou une suite finie d'éléments de E ou une permutation de E qui satisfait une certaine contrainte.

– A chaque solution S est associée une fonction objectif $f(S)$: on cherche donc une solution qui maximise cette fonction objectif.

3.3.6.2 Complexité des algorithmes gloutons

Soit n le cardinal de E. La complexité est donc souvent de l'ordre de $n \log n$ (le tri selon le critère) $+n \times f(n)$, si $f(n)$ est le coût de la vérification de la contrainte et de l'ajout d'un élément. Donc la complexité est souvent de l'ordre $O(n \log n + n \times f(n))$. Les algorithmes gloutons sont donc en général efficaces.

Exemple 3.3.2 (Problème de monnayeur)

Lorsque vous passez à la caisse d'un magasin quelconque, il n'est pas rare que le caissier doit vous rendre de l'argent, car le montant que vous lui avez donné est supérieur à celui que vous devez payer. Soient :

– s entier, c'est la somme à rendre ;
– E, ce sont les valeurs des pièces de monnaie dont on dispose, $E = \{e_1, ..., e_n\}$;
– Pour chaque valeur e_i, le nombre de pièces (ne_i) est non borné.

L'objectif est de constituer la somme s avec le moins possible de pièces.
Prenons $E = \{1, 2, 5, 10, 20, 50, 100, 200\}$ et $s = 263$ dinars.
Une solution $S = (ne_1, ..., ne_n)$ présente le nombre de pièces (ne_i) pour chaque valeur (e_i)

– Elle est correcte si $\sum_{i=1}^{n}(ne_i \times e_i) = s$.
– Elle est optimale si $\sum_{i=1}^{n}(ne_i)$ est minimal.

Algorithme glouton

Trier (E) (en ordre décroissant) ;
Init (S) (initialiser la solution à 0 ($ne_i = 0$)) ;
$i \leftarrow 1$;
Tant que ($i \leq n$) **faire**
    $S[i] \leftarrow s \; div \; E[i]$;
    $s \leftarrow s \quad mod \quad E[i]$;
    $i++$;
Fin tant que

Dans notre exemple la somme à rendre est 263 DA. En appliquant l'algorithme glouton, on aura la solution suivante :

- 1 pièce de 200 dinars ;
- 1 pièce de 50 dinars ;
- 1 pièce de 10 dinars ;
- 1 pièce de 2 dinars ;
- 1 pièce de 1 dinar.

Exemple 3.3.3 (Sac-à-dos)

Reprenons l'exemple (2.2.1) cité dans la page 14 :

$$(P)\begin{cases} max\ z = 7x_1 + 3x_2 + 4x_3 + 3x_4, \\ s.c \quad 13x_1 + 8x_2 + 12x_3 + 10x_4 \leq 9, \\ \qquad x_j \in \{0,1\}, j = 1, ..., 4. \end{cases}$$

On a trois choix gloutons possibles

- choisir l'objet le plus léger,
- choisir l'objet de plus grande valeur,
- choisir l'objet dont le rapport valeur/poids est maximale.

E	1	2	3	4
P	13	8	12	10
V	7	3	4	3
V/P	7/13	3/8	1/3	3/10

Par rapport au premier choix glouton l'objet, le plus léger est x_2, il sera alors mis dans le sac. La capacité restante est égale à 1, cela implique qu'aucun ajout d'un autre objet n'est possible.

pour le second choix, l'objet de plus grande valeur avec un poids qui dépasse pas la capacité du sac est x_2, aucun ajout d'un autre objet n'est possible.

Le rapport V/P maximal correspond à x_1, mais son poids dépasse la capacité du sac, alors il ne sera pas pris, le deuxième objet dont le rapport V/P est maximal est x_2 qui ne dépasse pas la capacité, alors il sera mis, aucun ajout d'un autre objet n'est possible.

	W=9
L'objet le plus léger	P=8, avec V=3
L'objet de plus grande valeur	V=3, avec P=8
L'objet dont le rapport V/P est maximal	V/P=3/8, avec V=3

Les trois choix gloutons nous ont donné la même solution qui est $x = (0, 1, 0, 0)$ avec une utilité de 3.

Exemple 3.3.4 (Voyageur de commerce [14])

Wolsey applique un algorithme glouton qui construit arc par arc le tour. Cette heuristique examine les arcs en les ordonnant par ordre croissant de longueur.
considérons le problème de TSP-symétrique avec 6 villes où la ville 1 est la ville de départ :

$$\begin{pmatrix} - & 9 & 2 & 8 & 12 & 11 \\ 9 & - & 7 & 19 & 10 & 32 \\ 2 & 7 & - & 29 & 18 & 6 \\ 8 & 19 & 29 & - & 24 & 3 \\ 12 & 10 & 18 & 24 & - & 19 \\ 11 & 32 & 6 & 3 & 19 & - \end{pmatrix}$$

L'arc ayant une longueur minimale est $e_1 = (1, 3)$ avec $c_{e_1} = 2$. On sélectionne cet arc en posant $x_{e_1} = 1$.

Le prochain arc avec une longueur minimale est $e_2 = (4, 6)$ avec $c_{e_2} = 3$ et $x_{e_2} = 1$.

Posons $x_{e_3} = 1$ tel que $e_3 = (3, 6)$ et $c_{e_3} = 6$.

Posons $x_{e_4} = 0$ tel que $e_4 = (2, 3)$ parce que le nœud 3 a été déjà visité, ainsi ces trois arcs : (1,3), (3,6), (2,3) ne peuvent pas être dans la tournée.

Posons $x_{e_5} = 0$ tel que $e_5 = (1, 4)$ car cet arc forme un cycle avec les arcs déjà choisis.

Posons $x_{e_6} = 1$ tel que $e_6 = (1, 2)$ avec $c_{e_6} = 9$.

Enfin, choisir les arcs (4,2) de longueur 19 et (2,5) de longueur 10 pour compléter le tour.

La tournée gloutonne trouvée est (1,3,6,4,2,5,1) avec une longueur de $\sum_e c_e x_e = 52$.

3.4 Conclusion

Dans ce chapitre, nous avons donné un aperçu sur les méthodes de résolution heuristiques qui sont utilisées dans le domaine de la programmation linéaire en nombres entiers. Plusieurs raisons de leur utilisation ont été citées et une grande variété de méthodes ont été présentées.

Chapitre 4

Méthode hybride pour la résolution des problèmes de PLE

4.1 Introduction

Après avoir vu les méthodes exactes ainsi que les méthodes approchées, une dernière classe d'algorithmes peut être trouvée dans la littérature pour résoudre des problèmes de PLE. C'est le principe d'hybridation des méthodes. En effet, il est possible d'exploiter plusieurs méthodes pour en former une nouvelle. Dans ce cas, nous obtenons une méthode dite « hybride ».

Le principe consiste alors à combiner des algorithmes exacts et/ou des algorithmes approchés pour essayer de tirer profit des points forts de chaque approche et améliorer le comportement global de l'algorithme. Une méthode hybride peut être construite par une combinaison de deux ou plusieurs méthodes exactes telles que l'algorithme de Marsten et Morin [41], proposé pour la résolution de problèmes mathématiques et qui combine la méthode de séparation et évaluation et la programmation dynamique, cette méthode a été appliquée au problème du sac-à-dos par Toth [42] et a permis l'obtention de bons résultats. On peut aussi combiner une méthode exacte avec une heuristique. Pour ce cas, plusieurs combinaisons ont été faites comme l'algorithme récemment proposé par Sörensen et Sevaux [43] appelé MA|PM pour (Memetic Algorithm and Population Management) selon la terminologie anglaise. Un algorithme mémétic, encore appelé algorithme génétique hybride, combine la recherche locale avec des algorithmes génétiques. Une autre méthode assez connue est l'algorithme GRASP (Greedy Randomized Adaptative Search Procedure). Il s'agit d'une procédure itérative qui peut être vue comme une approche hybride. Elle combine en effet les principes des heuristiques gloutonnes, de la recherche aléatoire et des méthodes de voisinage. Différentes méthodes hybrides très efficaces ont aussi été recensées et il nous semble que ce genre d'approches soit le plus à même de nos jours de permettre d'avoir des processus de résolution efficaces.

Dans ce chapitre, nous présenterons une méthode hybride obtenue en créant en com-

binant la méthode de support à variables bornées pour les problèmes de PLE et une heuristique basée sur des arrondis, puis nous passerons à une application sur une instance de problèmes possédant 10 variables pour pouvoir juger de la qualité de la solution qui sera obtenue pour un tel type de problème.

4.2 Présentation de la méthode hybride

Combiner entre deux méthodes n'est pas une chose facile à réaliser. D'abord, il faut savoir pour quelle raison on crée cette hybridation. Puis, les questions qui se poseront sont : Comment combiner ? Quel objectif veut-on atteindre ?. Cela se fait par utiliser les avantages de chacune des méthodes considérées.

Dans cette section, nous présentons une méthode hybride en combinant entre la méthode de support et l'heuristique d'arrondissement que l'on va définir dans la suite.

L'avantage que présente la méthode de support pour résoudre un problème de PLE, est qu'on est sûr d'obtenir une solution optimale, car elle parcourt tous les points du domaine admissible (le maximum peut être atteint en tout point du polyèdre), mais vu le nombre important d'itérations effectuées, cela induit un temps d'exécution très grand et le fait d'utiliser les coupes, la solution entière ne sera trouvée qu'à la fin. Pour cela, associer une heuristique d'arrondissement à la méthode de support s'avère très utile pour le processus de recherche d'une solution entière optimale.

Considérons le problème de programmation linéaire en nombres entiers et à variables bornées, s'écrivant sous la forme canonique suivante :

$$(PLE)\begin{cases} max\ z = c'x, \\ s.c \quad Ax \leq b, \\ \qquad d^- \leq x \leq d^+,\ x \text{ entier.} \end{cases} \tag{4.1}$$

On considère alors le problème relaxé standard associé à ce problème comme suit :

$$(PLR)\begin{cases} max\ z = c'x, \\ s.c \quad Ax + x_s = b, \\ \qquad d^- \leq x \leq d^+, x_s \in \mathbb{R}. \end{cases} \tag{4.2}$$

Nous pouvons résumer cette méthode hybride comme suit :

- Phase 1 : **Initialisation.** Choisir un plan de support initial $\{x, J_B\}$ du problème relaxé (4.2).
- Phase 2 : **Résolution du problème relaxé par la méthode de support à variables bornées.** Si la solution optimale obtenue est entière, on s'arrête. Sinon on passe à la phase 3.
- Phase 3 : **Heuristique d'arrondissement.** Arrondir la solution obtenue dans la phase 2 selon une procédure d'arrondissement adéquate.
- Phase 4 : **Ajout de coupe valide.** Calculer la coupe à ajouter au problème (4.2) selon la procédure présentée dans la section (2.6.5) et utiliser la solution arrondie pour le calcul de nouveau plan de support initial du problème augmenté et reprendre à partir de la phase 2.

4.2.1 Heuristique d'arrondissement

Soit $y^0 = (x^0, x_s^0)$ une solution optimale du problème (4.2) standardisé, où x^0 est une solution optimale non entière et $x_s^0 = b - Ax^0$.
On pose alors $x_j^0 = [x_j^0] + \alpha_j, avec 0 \leq \alpha_j < 1, j \in J = \{1, 2, ..., n\}$,
où $[x_j^0]$ est la partie entière et $\alpha_j = \{x_j^0\}$ la partie fractionnaire.

Afin d'obtenir une solution réalisable entière pour le problème (4.1), un processus d'arrondissement a été proposé et qui est décrit comme suit :

On pose $M = z(x^0)$ et $m = z(\hat{x})$, où $\hat{y} = (\hat{x}, \hat{x_s})$ vérifie $z(\hat{x}) = \min_{y \in S} z(x)$.
On définit les ensembles d'indices suivant :
$J_c^+ = \{j \in J : c_j \geq 0\}$ et $J_c^- = \{j \in J : c_j < 0\}$.

1. Arrondi de x^0
   - (A_1) : Arrondi x^1 par rapport à la variable tel que :

$$x^1 = (x_j^1, j \in J), \qquad x_j^1 = \begin{cases} [x_j^0], & \text{si } 0 \leq \alpha_j^0 \leq \frac{1}{2}, \\ [x_j^0] + 1, & \text{si } \frac{1}{2} < \alpha_j^0 < 1. \end{cases}$$

Si $z(x^1) > z(x^0)$ alors x^1 n'est pas réalisable ;

Sinon, on vérifie alors l'admissibilité de la solution.

Si x^1 n'est pas réalisable, on fait l'arrondissement par rapport à la fonction z.

– (A_2) : Arrondi x^2 par rapport à la fonction z tel que :

$$x^2 = \begin{cases} [x^0], & \text{si } z(\alpha_j^0) = \sum\limits_{j\in J} c_j\alpha_j^0 \geq 0, \\ [x^0] + e, & \text{si } z(e) = \sum\limits_{j\in J} c_j \leq z(\alpha_j^0) < 0, \end{cases}$$

où $e = (1, 1, ..., 1) \in \mathbb{R}^n$.

Dans le cas où cet arrondi n'est pas réalisable, on pose :

$$x^2 = (x_j^2,\ j \in J), \quad x_j^2 = \begin{cases} [x_j^0], & j \in J_c^+, \\ [x_j^0] + 1, & j \in J_c^-. \end{cases}$$

Si x^2 n'est pas réalisable, on fait l'arrondissement du milieu x^m.

2. (A_3) : Arrondi du milieu x^m tel que $x^m = \frac{1}{2}(x^0 + \hat{x})$ et on arrondit x^m à x^3 comme suit :

$$x^3 = (x_j^3,\ j \in J), \qquad x_j^3 = \begin{cases} [x_j^m] + 1, & j \in J_c^+, \\ [x_j^m], & j \in J_c^-. \end{cases}$$

Si cet arrondi n'est pas admissible, on pose :

$$x^3 = \begin{cases} [x^m] + 1, & \text{si } \mid J_c^+ \mid \geq \mid J_c^- \mid, \\ [x^m], & \text{si } \mid J_c^+ \mid < \mid J_c^- \mid . \end{cases}$$

Enfin, si cet arrondi n'est pas admissible, on pose

$$x^3 = (x_j^3,\ j \in J), \qquad x_j^3 = \begin{cases} [x_j^m], & j \in J_c^+, \\ [x_j^m] + 1, & J \in J_c^-. \end{cases}$$

Remarque 4.2.1 *L'arrondi x^3 a plus de probabilité à être réalisable, car x^m se situe au milieu du polyèdre S ou au milieu de l'une de ses faces.*

Remarque 4.2.2 *Même si à la fin de l'étape d'arrondissement, on n'a pas pu obtenir un arrondi réalisable, on continue l'algorithme des coupes jusqu'à l'obtention d'un arrondi réalisable satisfaisant ou de la solution optimale entière.*

4.2.2 Estimation de β-optimalité

Après avoir appliqué l'heuristique d'arrondissement, on peut évaluer la qualité de la solution réalisable entière x^{0a} trouvée par le calcul de l'estimation de β-optimalité comme suit :

$$\beta(x^{0a}) = \frac{z(x^0) - z(x^{0a})}{M - m}, \tag{4.3}$$

où M et m sont définis précédemment.
Comme la différence entre la valeur optimale $z(x^0)$ et la valeur $z(x^{0e})$, ne peut pas être aussi petite que l'on veut, la différence $z(x^0) - z(x^{0a})$ est divisée par la quantité $M - m$, de sorte qu'on aura :

$$0 \leq \frac{z(x^{0e}) - z(x^{0a})}{M - m} \leq \frac{z(x^0) - z(x^{0a})}{M - m} = \beta(x^{0a}) < 1.$$

Si $\beta(x^{0a}) \leq \beta$, où β est choisi à l'avance comme précision, alors on arrête. Sinon, on doit générer une coupe valide en déterminant le plan de support $\{\bar{y}, \bar{J}_B\}$.

Proposition 4.2.1 *Soit $\{y^0 = (x^0, x_s^0), J_B^0\}$ une solution optimale de support du problème (4.2), où x^0 et non entier et il existe $i_1 \in J_B^0$ tel que $Z_{i1}(y^0) < 0$.*
Soit $y^* = (x^*, x_s^*)$ une solution réalisable du problème (4.2) augmenté de la coupe $Z_{i_1}(y) \geq 0$. Alors le couple $(\bar{y}, \bar{J}_B)$ où

$$\bar{y} = y^* + \lambda(y^0 - y^*) \text{ et } \bar{J}_B = J_B^0 \cup \{n + m + 1\},$$

est une solution réalisable de support pour le problème (4.2) augmenté.

Rappelons que le paramètre λ est tel que $Z_{i_1}(\bar{y}) = 0$ et $0 \leq \lambda < 1$.

$$\lambda = \frac{Z_{i_1}(y^*)}{Z_{i_1}(y^*) - Z_{i_1}(y^0)}.$$

Schéma de l'algorithme

ENTRÉE : Une instance d'un problème de PLE ;

SORTIE : Une solution optimale x^0 ou approchée x^{0a}.

Etape 1 Soit $\{y = (x, x_s), J_B\}$ une solution réalisable initiale de support du problème (4.2). On construit alors un solution de support optimale $\{y^0 = (x^0, x_s^0), J_B^0\}$ par la méthode de support à variables bornées.

Si $x^0 \in \mathbb{N}^n$, arrêter l'algorithme ; x^0 est la solution optimale du problème (4.1).

Sinon, aller à l'étape 2.

Etape 2 Appliquer la procédure d'arrondissement décrite ci-dessus.

Soit x^{0a} l'arrondi obtenu.

Si x^{0a} est admissible, on calcule l'estimation de β-optimalité selon la formule (4.3).

Si $\beta(x^{0a}) \leq \beta$, on arrête, x^{0a} est β-optimal.

Sinon Poser $x^* = x^{0a}$, aller à l'étape 3.

Sinon On pose $x^* = x^m$ et aller à l'étape 3.

Etape 3 On construit la coupe $Z_{i_1}(y) \geq 0$.

Etape 4 Résoudre le problème augmenté suivant :

$$\begin{cases} max\ z = c'x, \\ \quad Ax + x_s = b, \\ \quad Z_{i_1}(x) \geq 0, \\ \quad d^- \leq x \leq d^+. \end{cases} \tag{4.4}$$

On commence par la solution réalisable de support $\{\bar{y}, \bar{J}_B\}$, avec :

$$\bar{y} = y^* + \lambda(y^0 - y^*),\ \ 0 \leq \lambda < 1,\ \ y^* = (x^*, x_s^*),\ \ x_s^* = b - Ax^*.$$

4.3 Application de la méthode sur un exemple

Pour appliquer la méthode hybride présentée, nous avons pris un exemple dans une thèse de doctorat [35]. Cet exemple est une instance du problème de sac-à-dos multidimensionnel en variables 0-1 qui est une généralisation du problème de sac-à-dos, où $m > 1$. Il a été résolu par une méthode hybride qui conjugue la programmation dynamique et la métaheuristique « recherche tabou ». Il a abouti à la solution optimale suivante :

$$x^* = (1, 0, 0, 1, 0, 1, 1, 0, 0, 0)', \text{ avec } z = (x^*) = 50.$$

Notre objectif est de trouver une solution la plus proche possible de x^*.
Soit à résoudre le problème de PLE suivant :

$$\begin{cases} max\ z(x) = 20x_1 + 18x_2 + 15x_3 + 14x_4 + 12x_5 + 9x_6 + 7x_7 + 5x_8 + 3x_9 + 2x_{10}, \\ 15x_1 + 16x_2 + 12x_3 + 12x_4 + 10x_5 + 10x_6 + 8x_7 + 5x_8 + 4x_9 + 3x_{10} \leq 45, \\ 22x_1 + 21x_2 + 16x_3 + 14x_4 + 15x_5 + 7x_6 + 5x_7 + 2x_8 + 4x_9 + 4x_{10} \leq 50, \\ 18x_1 + 20x_2 + 15x_3 + 10x_4 + 9x_5 + 8x_6 + 2x_7 + 6x_8 + 2x_9 + 5x_{10} \leq 40, \\ 0 \leq x_i \leq 1,\ x_i \text{ entier},\ i = \overline{1.10}. \end{cases} \tag{4.5}$$

Le problème relaxé associé s'écrit sous forme standard :

$$\begin{cases} max\ z(x) = 20x_1 + 18x_2 + 15x_3 + 14x_4 + 12x_5 + 9x_6 + 7x_7 + 5x_8 + 3x_9 + 2x_{10}, \\ 15x_1 + 16x_2 + 12x_3 + 12x_4 + 10x_5 + 10x_6 + 8x_7 + 5x_8 + 4x_9 + 3x_{10} + x_{11} = 45, \\ 22x_1 + 21x_2 + 16x_3 + 14x_4 + 15x_5 + 7x_6 + 5x_7 + 2x_8 + 4x_9 + 4x_{10} + x_{12} = 50, \\ 18x_1 + 20x_2 + 15x_3 + 10x_4 + 9x_5 + 8x_6 + 2x_7 + 6x_8 + 2x_9 + 5x_{10} + x_{13} = 40, \\ 0 \leq x_i \leq 1,\ i = \overline{1.10},\ 0 \leq x_{11} \leq 45,\ 0 \leq x_{12} \leq 50,\ 0 \leq x_{13} \leq 40. \end{cases} \tag{4.6}$$

Étape 1 : Résoudre par la méthode directe de support le problème relaxé (4.6).
Soit $x = (0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 15, 28, 17)'$ et $J_B = \{8, 9, 10\}$ un plan initial de support du problème (4.6) avec $z(x) = 26$.
Après 15 itérations, nous avons trouvé le plan de support optimal suivant :

$$y^0 = (x^0, x_s^0) = (\frac{66}{73}, 0, 0, 1, \frac{42}{73}, \frac{5}{73}, 1, 1, 0, 0, 0, 0, 0)',\ J_B^0 = \{6, 1, 5\},\ M = z(y^0) = \frac{3767}{73} \simeq 51.602.$$

De plus, on a $\hat{y} = (\hat{x}, \hat{x_s}) = (0,0,0,0,0,0,0,0,0,0,45,50,40)'$, $m = z(\hat{y}) = 0$. Le vecteur $x^0 = (\frac{66}{73}, 0, 0, 1, \frac{42}{73}, \frac{5}{73}, 1, 1, 0, 0)'$ étant non entier, on applique la procédure d'arrondissement sur x^0.

Premier arrondi : Arrondi x^1 par rapport à la variable :

$x^1 = (1,0,0,1,1,0,1,1,0,1)'$ avec $z(x^1) = 58$, cette solution n'est pas réalisable.

deuxième arrondi : Arrondi x^2 par rapport à la fonction $z(x)$:

$x^2 = (0,0,0,1,0,0,1,1,0,0)'$ cette solution est réalisable.

Alors, $x^{0a} = x^2$ et $x_s^{0a} = (20,29,22)'$,

d'où $y^{0a} = (x^{0a}, x_s^{0a}) = (0,0,0,1,0,0,1,1,0,0,20,29,22)'$ et $z(x^{0a}) = z(y^{0a}) = 26$.

L'estimation de β-optimalité vaut :

$$\beta(x^{0a}) = \frac{1869}{3767} \simeq 0.497.$$

L'arrondi x^{0a} n'étant pas satisfaisant, on pose alors $x^* = x^{0a}$, $y^{0a} = (x^{0a}, x_s^{0a})$.

Étape 2 : On ajoute au programme (4.6) une coupe.

Itération 1 Le vecteur des estimations vaut :

$E_N^0 = (E_2^0, E_3^0, E_4^0, E_7^0, E_8^0, E_9^0, E_{10}^0, E_{11}^0, E_{12}^0, E_{13}^0) = (\frac{1073}{365}, \frac{286}{365}, \frac{-454}{365}, \frac{-766}{365}, \frac{-1}{73}, \frac{147}{365}, \frac{929}{365}, \frac{108}{365}, \frac{23}{73}, \frac{35}{73})'$,

$J_N^0 = \{2,3,4,7,8,9,10,11,12,13\}$.

On partitionne l'ensemble J_N^0 comme suit :

$J_N^+ = \{2,3,9,10,11,12,13\}$,

$J_N^- = \{4,7,8\}$,

$J^+ = J^- = \emptyset$.

On calcule alors les fonctions $Z_i(y^0)$, $i \in J_B^0$:

on trouve

$$Z_B(y^0) = -\left\{ \begin{pmatrix} 158/73 \\ 27/73 \\ 130/73 \end{pmatrix} - \begin{pmatrix} 184/365 & 341/365 & 48/73 \\ -24/365 & -346/364 & 35/73 \\ 58/73 & 94/73 & -64/73 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right\} = (-5/73, -66/73, -42/73)'.$$

Alors on ajoute au système (4.6) la contrainte $Z_{i_1}(y^0) \geq 0$ tel que $i_1 = 2$.

$$Z_{i_1}(y) = \frac{-66}{73} + (\frac{113}{365}, \frac{113}{365}, \frac{257}{365}, \frac{189}{365}, \frac{308}{365}, \frac{71}{73}, \frac{16}{73}) \begin{pmatrix} x_2 \\ x_3 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \end{pmatrix} + (\frac{24}{365}, \frac{346}{365}, \frac{113}{365}, \frac{38}{73}) \begin{pmatrix} -x_4+1 \\ -x_7+1 \\ -x_8+1 \end{pmatrix} \geq 0.$$

D'où la contrainte suivante :

$113x_2 + 356x_3 - 24x_4 - 346x_7 - 190x_8 + 257x_9 + 189x_{10} + 308x_{11} + 355x_{12} + 80x_{13} - x_{14} = -230.$

Le problème augmenté est :

$$\begin{cases} max\ z(x) = 20x_1 + 18x_2 + 15x_3 + 14x_4 + 12x_5 + 9x_6 + 7x_7 + 5x_8 + 3x_9 + 2x_{10}, \\ 15x_1 + 16x_2 + 12x_3 + 12x_4 + 10x_5 + 10x_6 + 8x_7 + 5x_8 + 4x_9 + 3x_{10} + x_{11} = 45, \\ 22x_1 + 21x_2 + 16x_3 + 14x_4 + 15x_5 + 7x_6 + 5x_7 + 2x_8 + 4x_9 + 4x_{10} + x_{12} = 50, \\ 18x_1 + 20x_2 + 15x_3 + 10x_4 + 9x_5 + 8x_6 + 2x_7 + 6x_8 + 2x_9 + 5x_{10} + x_{13} = 40, \\ 113x_2 + 356x_3 - 24x_4 - 346x_7 - 190x_8 + 257x_9 + 189x_{10} + 308x_{11} + 355x_{12} + 80x_{13} - x_{14} = -230, \\ 0 \leq x_i \leq 1,\ i = \overline{1.10},\ 0 \leq x_{11} \leq 45,\ 0 \leq x_{12} \leq 50,\ 0 \leq x_{13} \leq 40,\ 0 \leq x_{14} \leq 35955. \end{cases} \tag{4.7}$$

On résout le programme (4.7) en commençant par la solution initiale suivante :

$$\bar{y} = (\bar{x}, \bar{x_s}) = y^* + \lambda(y^0 - y^*),$$

où $\lambda = \frac{3577}{3643}$,

donc

$$\bar{y} = (\frac{3234}{3643}, 0, 0, 1, \frac{2058}{3643}, \frac{245}{3643}, 1, 1, 0, 0, \frac{1320}{3643}, \frac{1914}{3643}, \frac{1452}{3643})'.$$

Après l'ajout de la variable d'écart x_{14},

$$\bar{y} = (\frac{3234}{3643}, 0, 0, 1, \frac{2058}{3643}, \frac{245}{3643}, 1, 1, 0, 0, \frac{1320}{3643}, \frac{1914}{3643}, \frac{1452}{3643}, 0)',$$

et $\bar{J_B} = \{6, 1, 5, 14\}$.

Après 4 itérations, nous avons obtenu une solution optimale de support qui est :

$$y^1 = (1, 0, 0, 1, \frac{10}{33}, \frac{3}{11}, 1, \frac{28}{33}, 0, 0, 0, \frac{28}{33}, 0, 0)',\ J_B^1 = \{6, 8, 5, 12\},\quad \text{avec } Z(y^1) = \frac{154}{3} \simeq 51.333.$$

Itération 2 : Le vecteur obtenu $x^1 = (1, 0, 0, 1, \frac{10}{33}, \frac{3}{11}, 1, \frac{28}{33}, 0, 0)'$ n'étant pas entier, on applique la procédure d'arrondissement sur x^1, où le premier arrondi par rapport à la variable est réalisable. On aura alors $x^{1a} = (1, 0, 0, 1, 0, 0, 1, 1, 0, 0)'$, $x_s^{1a} = (5, 7, 4, 4015)'$, avec $z(x^{1a}) = 46$.

L'estimation de β-optimalité vaut :

$$\beta(x^{1a}) = \frac{16}{154} \simeq 0.103.$$

L'arrondi x^{1a} n'étant pas satisfaisant, on pose alors $x^* = x^{1a}$, $y^* = (x^{1a}, x_s^{1a})$ et on ajoute une nouvelle coupe au programme (4.7).

Le vecteur des estimations vaut :

$E_N^1 = (E_1^1, E_2^1, E_3^1, E_4^1, E_7^1, E_9^1, E_{10}^1, E_{11}^1, E_{13}^1, E_{14}^1) = (\frac{-1}{3}, \frac{12}{5}, \frac{2}{15}, \frac{-6}{5}, \frac{-22}{15}, \frac{4}{15}, \frac{11}{5}, \frac{1}{15}, \frac{1}{3}, \frac{1}{1095})'$,

$J_N^1 = \{1, 2, 3, 4, 7, 9, 10, 11, 13, 14\}$.

On partitionne l'ensemble J_N^1 comme suit :

$J_N^+ = \{2, 3, 9, 10, 11, 13, 14\}$,

$J_N^- = \{1, 4, 7\}$,

$J^+ = J^- = \emptyset$.

On calcule alors le vecteur des fonctions $Z_i(y^1)$, $i \in J_B^1$:

on trouve

$$Z_B(y^1) = -\left\{ \begin{pmatrix} 18/11 \\ 25/33 \\ 82/33 \\ -8/33 \end{pmatrix} - \begin{pmatrix} -14/11 & 32/55 & 113/55 \\ 71/33 & -8/55 & -346/165 \\ 56/33 & 38/55 & -34/165 \\ 38/33 & -8/55 & -364/165 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right\} = (-3/11, -28/33, -10/33, -28/33)'.$$

Alors on ajoute au système (4.7) la contrainte $Z_i(y^1) \geq 0$ tel que $i_1 = 2$.

$$Z_{i_1}(y) = Z_2(y) = \frac{-28}{33} + (\frac{46}{55}, \frac{26}{165}, \frac{67}{165}, \frac{8}{55}, \frac{118}{165}, \frac{16}{33}, \frac{12043}{12045}) \begin{pmatrix} x_2 \\ x_3 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{13} \\ x_{14} \end{pmatrix} + (\frac{28}{33}, \frac{8}{55}, \frac{16}{165}) \begin{pmatrix} -x_1 + 1 \\ -x_4 + 1 \\ -x_7 + 1 \end{pmatrix} \geq 0.$$

D'où la contrainte suivante :

$$-10220x_1 + 10074x_2 + 1898x_3 - 1752x_4 - 1168x_7 + 4891x_9 + 1752x_{10} + 8614x_{11} + 5840x_{13} + 12043x_{14} - x_{15} = -2920.$$

Le problème augmenté est :

$$\begin{cases} max\ z(x) = 20x_1 + 18x_2 + 15x_3 + 14x_4 + 12x_5 + 9x_6 + 7x_7 + 5x_8 + 3x_9 + 2x_{10}, \\ 15x_1 + 16x_2 + 12x_3 + 12x_4 + 10x_5 + 10x_6 + 8x_7 + 5x_8 + 4x_9 + 3x_{10} + x_{11} = 45, \\ 22x_1 + 21x_2 + 16x_3 + 14x_4 + 15x_5 + 7x_6 + 5x_7 + 2x_8 + 4x_9 + 4x_{10} + x_{12} = 50, \\ 18x_1 + 20x_2 + 15x_3 + 10x_4 + 9x_5 + 8x_6 + 2x_7 + 6x_8 + 2x_9 + 5x_{10} + x_{13} = 40, \\ 113x_2 + 356x_3 - 24x_4 - 346x_7 - 190x_8 + 257x_9 + 189x_{10} + 308x_{11} + 355x_{12} + 80x_{13} - x_{14} = -230, \\ -10220x_1 + 10074x_2 + 1898x_3 - 1752x_4 - 1168x_7 + 4891x_9 + 1752x_{10} + 8614x_{11} + 5840x_{13} + 12043x_{14} - x_{15} = -2920, \\ 0 \leq x_i \leq 1,\ i = \overline{1.10},\ 0 \leq x_{11} \leq 45,\ 0 \leq x_{12} \leq 50,\ 0 \leq x_{13} \leq 40,\ 0 \leq x_{14} \leq 35955,\ 0 \leq x_{15} \leq 433648830. \end{cases} \tag{4.8}$$

On résout le programme (4.8) en commençant par la solution initiale suivante :

$$\bar{y^1} = (\bar{x^1}, \bar{x_s^1}) = y^* + \lambda(y^1 - y^*),$$

où $\lambda = \frac{132627}{132655}$,

donc

$$\bar{y} = (1, 0, 0, 1, \frac{8038}{26531}, \frac{36171}{132655}, 1, \frac{22512}{26531}, 0, 0, \frac{28}{26531}, \frac{112728}{132655}, \frac{112}{132655}, \frac{22484}{26531})'.$$

Après l'ajout de la variable d'écart x_{15},

$$\bar{y} = (1, 0, 0, 1, \frac{8038}{26531}, \frac{36171}{132655}, 1, \frac{22512}{26531}, 0, 0, \frac{28}{26531}, \frac{112728}{132655}, \frac{112}{132655}, \frac{22484}{26531}, 0)',$$

et $\bar{J_B^1} = \{6, 8, 5, 12, 15\}$.

Après 3 itérations, nous avons obtenu une solution optimale de support qui est :

$$y^2 = (1, 0, 0, 1, \frac{3646}{12043}, \frac{3287}{12043}, 1, \frac{10220}{12043}, 0, 0, 0, \frac{10248}{12043}, 0, \frac{10220}{12043}, 0)',\ J_B^2 = \{6, 8, 5, 12, 14\},$$

avec $z(y^2) = \frac{618198}{12043} \simeq 51.332$.

Itération 3 : Le vecteur obtenu $x^2 = (1, 0, 0, 1, \frac{3646}{12043}, \frac{3287}{12043}, 1, \frac{10220}{12043}, 0, 0)'$ n'étant pas entier, on applique la procédure d'arrondissement sur x^2, où le premier arrondi par rapport à la variable est réalisable. On aura alors $x^{2a} = (1, 0, 0, 1, 0, 0, 1, 1, 0, 0)'$,

$x_s^{2a} = (5, 7, 4, 4015, 48408855)'$, avec $z(x^{2a}) = 46$.

L'estimation de β-optimalité vaut :

$$\beta(x^{2a}) = \frac{64220}{618195} \simeq 0.103.$$

L'arrondi x^{2a} n'étant pas satisfaisant, on pose alors $x^* = x^{2a}$, $y^* = (x^{2a}, x_s^{2a})$.

Le vecteur des estimations vaut :

$E_N^2 = (E_1^2, E_2^2, E_3^2, E_4^2, E_7^2, E_9^2, E_{10}^2, E_{11}^2, E_{13}^2, E_{15}^2)$

$= (\frac{-4005}{12043}, \frac{28894}{12043}, \frac{1604}{12043}, \frac{-14450}{12043}, \frac{-17662}{12043}, \frac{3207}{12043}, \frac{26493}{12043}, \frac{795}{12043}, \frac{4009}{12043}, \frac{1}{13187085})'$,

$J_N^2 = \{1, 2, 3, 4, 7, 9, 10, 11, 13, 15\}$.

On partitionne l'ensemble J_N^2 comme suit :

$J_N^+ = \{2, 3, 9, 10, 11, 13, 15\}$,

$J_N^- = \{1, 4, 7\}$,

$J^+ = J^- = \emptyset$.

On calcule alors le vecteur des fonctions $Z_i(y^2)$, $i \in J_B^2$:

on trouve

$$Z_B(y^2) = \left(\frac{3287}{12043}, \frac{-10220}{12043}, \frac{-3646}{12043}, \frac{-10248}{12043}, \frac{-10220}{12043}\right)'.$$

Alors on ajoute au système (4.8) la contrainte $Z_{i_1}(y^2) \geq 0$ tel que $i_1 = 4$.

$$Z_{i_1}(y) = Z_4(y) = \frac{-10248}{12043} + (\frac{50508}{60215}, \frac{9516}{60215}, \frac{24522}{60215}, \frac{8784}{60215}, \frac{43188}{60215}, \frac{5856}{12043}, \frac{29011580}{29011587}) \begin{pmatrix} x_2 \\ x_3 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{13} \\ x_{15} \end{pmatrix} + (\frac{10248}{12043}, \frac{8784}{60215}, \frac{5856}{60215}) \begin{pmatrix} -x_1 + 1 \\ -x_4 + 1 \\ -x_7 + 1 \end{pmatrix} \geq 0.$$

D'où la contrainte suivante :

$$\frac{-10248}{12043}x_1 + \frac{50508}{60215}x_2 + \frac{9516}{60215}x_3 - \frac{8784}{60215}x_4 - \frac{5856}{60215}x_7 + \frac{24522}{60215}x_9 + \frac{8784}{60215}x_{10} + \frac{43188}{60215}x_{11} + \frac{5856}{12043}x_{13} + \frac{29011580}{29011587}x_{15} - x_{16} = \frac{-2928}{12043}.$$

Le problème augmenté est :

$$\begin{cases} max\ z(x) = 20x_1 + 18x_2 + 15x_3 + 14x_4 + 12x_5 + 9x_6 + 7x_7 + 5x_8 + 3x_9 + 2x_{10}, \\ 15x_1 + 16x_2 + 12x_3 + 12x_4 + 10x_5 + 10x_6 + 8x_7 + 5x_8 + 4x_9 + 3x_{10} + x_{11} = 45, \\ 22x_1 + 21x_2 + 16x_3 + 14x_4 + 15x_5 + 7x_6 + 5x_7 + 2x_8 + 4x_9 + 4x_{10} + x_{12} = 50, \\ 18x_1 + 20x_2 + 15x_3 + 10x_4 + 9x_5 + 8x_6 + 2x_7 + 6x_8 + 2x_9 + 5x_{10} + x_{13} = 40, \\ 113x_2 + 356x_3 - 24x_4 - 346x_7 - 190x_8 + 257x_9 + 189x_{10} + 308x_{11} + 355x_{12} + 80x_{13} - x_{14} = -230, \\ -10220x_1 + 10074x_2 + 1898x_3 - 1752x_4 - 1168x_7 + 4891x_9 + 1752x_{10} + 8614x_{11} + 5840x_{13} + 12043x_{14} - x_{15} = -2920, \\ \frac{-10248}{12043}x_1 + \frac{50508}{60215}x_2 + \frac{9516}{60215}x_3 - \frac{8784}{60215}x_4 - \frac{5856}{60215}x_7 + \frac{24522}{60215}x_9 + \frac{8784}{60215}x_{10} + \frac{43188}{60215}x_{11} + \frac{5856}{12043}x_{13} + \frac{29011580}{29011587}x_{15} - x_{16} = \frac{-2928}{12043}, \\ 0 \leq x_i \leq 1,\ i = \overline{1.10},\ 0 \leq x_{11} \leq 45,\ 0 \leq x_{12} \leq 50,\ 0 \leq x_{13} \leq 40,\ 0 \leq x_{14} \leq 35955,\ 0 \leq x_{15} \leq 433648830,\ 0 \leq x_{16} \leq 2172264580. \end{cases} \tag{4.9}$$

On résout le programme (4.9) en commençant par la solution initiale suivante :

$$\bar{y^2} = (\bar{x^2}, \bar{x_s^2}) = y^* + \lambda(y^2 - y^*),$$

où $\lambda = \frac{72873469558}{72873470839}$,
donc

$$\bar{y} = (1, 0, 0, 1, \frac{22062332476}{72873470839}, \frac{19889985422}{72873470839}, 1, \frac{61842304601}{72873470839}, 0, 0, \frac{6405}{72873470839}, \frac{62011743255}{72873470839}, \frac{5124}{72873470839}, \frac{61847446535}{72873470839}, \frac{62011743255}{72873470839})'.$$

Après l'ajout de la variable d'écart x_{16}

$$\bar{y} = (1, 0, 0, 1, \frac{22062332476}{72873470839}, \frac{19889985422}{72873470839}, 1, \frac{61842304601}{72873470839}, 0, 0, \frac{6405}{72873470839}, \frac{62011743255}{72873470839}, \frac{5124}{72873470839}, \frac{61847446535}{72873470839}, \frac{62011743255}{72873470839}, 0)',$$

et $\bar{J_B^2} = \{6, 8, 5, 12, 14, 16\}$.

Après 3 itérations, nous avons obtenu une solution optimale de support qui est :

$$y^3 = (1, 0, 0, 1, \frac{10979014}{36264475}, \frac{9897977}{36264475}, 1, \frac{30774968}{36264475}, 0, 0, 0, \frac{6171858}{7252895}, 0, \frac{6155506}{7252895}, \frac{6171858}{7252895}, 0)',$$

$J_B^3 = \{6, 8, 5, 12, 14, 15\}$, avec $z(y^3) = \frac{1861548276}{36264475} \simeq 51.332$.

Comme nous le constatons, l'application de notre approche hybride nous donne la même valeur de x pour trois coupes ajoutées.
Notre solution approchée est $x = (1, 0, 0, 1, 0, 0, 1, 1, 0, 0)'$ avec $z(x) = 46$, et $\beta(x^{3a}) \simeq 0.103$, donc on peut s'arrêter à ce niveau.

L'exemple que nous avons pris a une particularité, tous les coefficients sont positifs. Alors, si on joue sur les coefficients de la fonction objectif en essayant de modifier les composantes de la solution approchée qui nous donne 46, le pas maximal qu'on puisse faire vaut 4. De ce fait, les seules composantes qu'on peut toucher sont x_9 et x_{10}.
Si on leur affecte une valeur de 1 pour chacune avec $z(x) = 51$, la solution correspondante

n'est pas réalisable.

Si on pose $x_9 = 0$ et $x_{10} = 1$, la solution ne sera pas réalisable.

Si on pose $x_9 = 1$, et $x_{10} = 0$, on aura une solution réalisable $\tilde{x} = (1, 0, 0, 1, 0, 0, 1, 1, 1, 0)'$, avec $z(\tilde{x}) = 49$ qui sera considérée comme la meilleure solution qu'on puisse atteindre si on se limite avec 3 coupes générées.

4.4 Conclusion

Nous avons présenté dans ce chapitre une approche hybride utilisant à la fois la méthode directe de support à variables bornées et l'heuristique d'arrondissement pour résoudre des problèmes en nombres entiers. La méthode de support est appliquée sur le problème relaxé pour avoir une solution optimale, ensuite un arrondissement sera appliqué sur cette solution. A l'aide de ces deux solutions, nous calculons une nouvelle solution initiale pour notre programme augmenté par la méthode de support.

Nous avons validé cette approche sur le problème du sac-à-dos multidimensionnel, et cela a donné une bonne solution. Si nous appliquons la méthode d'arrondissement sur l'exemple traité avec la méthode directe de support dans la section (2.6), nous obtenons directement la solution optimale sans passage au calcul des coupes.

Conclusion générale

Le développement rapide de la programmation linéaire en nombres entiers au cours des 50 dernières années peut être mesuré non seulement par le nombre important des publications qui lui ont été consacrées, mais aussi par l'amélioration des capacités des algorithmes de résolution, que ce soient les méthodes exactes ou les méthodes approchées et enfin, la meilleure idée plus que jamais est "l'hybridation de ces méthodes".

On a vu que les algorithmes des coupes ne donnent de solution optimale qu'à la fin, c'est-à-dire en un temps de calcul considérable. Par contre, les heuristiques essaient de trouver des solutions réalisables de bonne qualité en un temps raisonnable, mais sans garantir l'obtention d'une solution optimale.

Dans ce mémoire, lorsqu'on a combiné la méthode exacte avec l'heuristique proposée, on a vu que cette méthode hybride nous donne une bonne solution proche de l'optimum qui satisfait le critère de β-optimalité.

Bibliographie

[1] Moisdon. Jean-Claude, Nakhla Michel. Recherche opérationnelle : méthodes d'optimisation en gestion. Presses des Mines. Paris, 2015.

[2] Lilia Zaourar. Recherche opérationnelle et optimisation pour la conception testable de circuits intégrés complexes. Thèse de doctorat. Université de Grenoble, 2010.

[3] M.R. Garey and D.S. Johnson. Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical sciences. WH Freeman and Company, San Francisco, Calif,1979.

[4] CH Papadimitriou. Combinatorial Optimization : Algorithm and Complexity. Prentice-Hall, New York, 1982.

[5] O. Goldreich. Computanional complexity : a conceptual perspective. Cambridg University Press, 2008.

[6] Stanistaw Walukiewicz. Integer programming. Mathematics and its applications(Springer-Science+Business Media, B.V.). Varsovie,1990.

[7] M. W. Padberg, Ed. Combinatorial Optimization. Mathematical Programming Study. Vol 12, pp 78-107. New York, 1980.

[8] Little, J. D. C., K. G. Murty, D. W. Sweeney, C. Karel. An Algorithm for the Traveling Salesman Problem. Operations Research, 11, 1963, 972-989

[9] Hoffman, K. L., M. Padberg. Solving Airline Crew Scheduling Problems by Branch-and-Cut. Management Science, 39, 1993.

[10] Chen, D., R. G. Batson, Y. Dang. Applied Integer Programming : Modeling and Solution. John Wiley and Sons. New Jersey, 2010.

[11] D.E. Goldberg. Genetic Algorithms in Search an Optimization. Addison-Wesley, Boston. USA, 1989.

[12] Jean-François Scheid. Programmation linéaire :Méthodes et applications. Techniques de l'ingénieur. Paris, 2015.

[13] Egon Balas, Matteo Fishetti, Arrigo Zanette. A hard integer program made easy by lexicography. Mathematical Programming. University of Padova. Vol 135, pp 509-514, 2011.

[14] L. A. Wolsey. Integer programming. John Wiley & Sons. New York, 1998.

[15] Land, A. H., A. G. Doig. An Automatic Method for Solving Discrete Programming Problems. Econometrica, 28, pp 97-520, 1960.

[16] Little, J.D.C., K. G. Murty, D. W. Sweeney, C. Karel. An Algorithm for the Traveling Salesman Problem. Operations Research, Vol 11, pp 972-989, 1963.

[17] D.Werra, T.M.Liebling, J.G.Hêche. Recherche opérationnelle pour ingénieur I.

[18] H.BOUSSOUIRA. Méthode de support pour la résolution des problèmes de programmation linéaire en nombres entiers et mixtes. Thèse de magister, université de Béjaïa, 2009.

[19] MINOUX (M.). Programmation Mathématique : Théorie et Algorithmes. Deuxième édition Lavoisier, Paris (2008).

[20] Daniel THIEL. Recherche opérationnelle et management d'entreprise. Economica. Paris, 1995.

[21] Richard Bellman, Dynamic programming. Princeton landmarks in mathematics. New York, 1957.

[22] Dethoor Jean-Marc. Au-delà de la programmation dynamique. In : Revue économique, volume 20, pp 515-535, 1969.

[23] M. O. BIBI. Cours de programmation linéaire et quadratique en master 1. Université de Béjaïa, 2017.

[24] Krasimira Genova, Vassil Guliashki. Linear Integer Programming Methods and Approaches – A Survey. Cybernetics And Information Technologies. Volume 11. Sofia, 2011.

[25] E. A. Silver. An Overview of Heuristic Solution Methods. Journal of the Operational Research Society, vol 55, pp 936-956 2004.

[26] Woolsey RED et Swanson HS. Operations research for immediate applications. Harper and Row : New York, 1975.

[27] Pierre Fouilhoux. Méthodes heuristiques en Optimisation Combinatoire. Document 3/5 : Partie 1 Métaheuristiques, 2009-2010.

[28] Michael Jünger. Thomas Liebling. Denis Naddef. George Nemhauser. William Pulleyblank. Gerhard Reinelt. Giovanni Rinaldi. Laurence Wolsey. 50 years of integer programming 1958-2008. The Early Years to the State-of-the-Art. Springer Verlag Berlin Heidelberg, 2010.

[29] T. Achterberg. Constraint integer programming. Ph.D. thesis, ZIB, Berlin, 2007.

[30] Bolat, A., M. Savsar, and M. Al-Fawzan. "Algorithms for Real-time Scheduling of Jobs on Mixed Model Assembly Lines", Computer and Operations Research, Vol. 21, No. 5, pp.487-498, 1994.

[31] Ramirez-Beltran, N. and K. Aguilar-Ruggiero. "Application of an Heuristic Procedure to Solve Mixed-Integer Programming Problems". Computers and Industrial Engineering, Vol. 33, pp 16-43 1997.

[32] Walser, J. Integer Optimization by Local Search : A Domain-Independent Approach, Springer, New York,2008.

[33] Danna, E., E. Rothberg, and C. LePape. "Integrating Mixed Integer Programming and Local Search : A Case Study on Job-Shop Scheduling Problems". CP-AI-OR'03, 2003.

[34] Danna, E. "Integrating Local Search Techniques into Mixed Integer Programming", 40R. Vol 2, pp. 321-324, 2004.

[35] Christophe Wilbaut. Heuristiques hybrides pour la résolution de problèmes en varia.bles 0-1 mixtes. Thèse de doctorat. Université Valenciennes et du Hainaut-Cambrésis, 2006.

[36] Preescott-Gagnon, Eric. Méthodes hybrides basées sur la génération de colonnes pour des problèmes de tournées de Véhicules avec fenêtres de temps. Thèse de doctorat. École Polytechnique de Montréal, février 2011

[37] Cherfi Nawal. Méthodes de résolution hybrides knapsack. Thèse de doctorat. Université Paris I, novembre 2008.

[38] J.Shapiro. "Dynamic Programming Algorithms for the Integer Programming Problem I : The Integer Programming Problem Viewed as a Knapsack Type Problem", Operations Research, Vol. 16, pp. 103-121, 1968.

[39] P. Siarry. La méthode de récuit simulé : théorie et application. Automatique-productique informatique industrielle, 29(4-5) : 535-561, 1995.

[40] M. Dorigo. M. Birattari and T.Stutzle. Ant colony optimization. IEEE computational intelligence magazine, I(4) : 28-39, 2006.

[41] R. E. Marsten, T. L. Morin. A hybrid approach to discrete mathematical programming, Mathematical Programming, 14 : 21–40, 1978.

[42] P. Toth. Dynamic programming algorithms for the zero-one knapsack problem. Computing, 25 :29-45, 1980.

[43] K. Sörensen et M. Sevaux. MA|PM : memetic algorithms with population management. Computers and Operations Research, Volume 33, Issue 5 : 1214-1225, 2006.

[44] RL. Ackoff. Optimization + objectivity = opt out. Eur J Opl Res 1 : 1-7, 1977.

[45] H. Müller-Merbach . Heuristics and their design : a survey. Eur J Opl Res 8 : 1-23, 1981.

[46] F. Glover, et G. Kochenberger. Critical Event tabu Search for Multidimensional Knapsack Problems. Dans I.H. Osman et J.P. Kelly (éditeurs), Meta Heuristics : Theory and Applications, Kluwer Academic Publishers, pp 407 – 427, 1996.

[47] S.Hillier, J.Liebermab. Introduction to oprations researche. Ninth Edition. New York, 2010