République algérienne démocratique et populaire

Ministère de l'enseignement supérieur et de la recherche scientifique

Université mouloud mammeri de tizi-ouzous

Faculté des Sciences

Département de Mathématiques

Mémoire de fin de cycle

En vue d'obtention du diplôme de Master en mathématiques appliquée à la gestion

Thème :

Les méthodes de résolution des problèmes de programmation linéaire en nombre entiers

Réalisé par :

M. : DJOUNADI Hocine

Devant les membres de jury :

Président : Dr. GOUBI

Examinateur : Dr. OUBAKOUK

Encadré : Dr. LESLOUS

Promotion: 2019

Remerciement

Je remercie dieu le tout puissant de m'avoir donné la force, la volonté et le courage pour accomplir ce travail.

Je tiens à remercier mon encadreur Dr LESLOUS qui a dirigé mon travail, pour ses précieux conseils et ses orientations.

J'adresse également mes plus profonds remerciements à nos enseignants.

Mes vifs remerciements s'adressent aux membres de jury qui ont accepté d'examiner notre travail.

Enfin, je remercie toute personne ayant contribuée de près ou de loin à la réalisation de ce modeste travail.

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 des méthodes de résolution pour obtenir des solutions de bonne qualité en un temps raisonnable. On combine par exemple 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,

Table des matières

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 **[51]**.

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 1000000 ; 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 **[51].**

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 **[38].**

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.

L'objectif du travail présenté dans ce mémoire est de présenter les méthodes de résolution des problèmes de programmation linéaire en nombres entiers. Cette approche efficace combine la méthode directe de support pour obtenir des solutions de bonne qualité en un temps d'exécution 'raisonnable '

Ce travail s'articule autour de 4 chapitres : le premier est l'optimisation combinatoire, au deuxième la programmation linéaire en nombre entier, au troisième et au dernier chapitre 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.

Chapitre 1

OPTIMISATION COMBINATOIRE

1. Introduction

L'optimisation combinatoire appelée aussi optimisation discrète, est une branche de l'optimisation en mathématiques appliquées et en informatique, elle est également liée à la recherche opérationnelle, à l'algorithmique et à la théorie de la complexité. Une des raisons de son développement est liée au nombre considérable de problèmes qu'elle permet de traiter, aussi bien dans les domaines de l'ingénieur que dans les domaines de recherche, ce qui a fait surgir un large éventail de concepts qui continuent d'être enrichies par une bibliographie très abondante et très dense, qui traite divers problèmes d'optimisation ainsi que leurs méthodes de résolution.

Un problème d'optimisation combinatoire consiste à trouver la meilleure solution dans un ensemble discret de solutions appelé ensemble des solutions réalisables. En général, cet ensemble est fini mais de cardinalité très grande **[1]**.

Généralement, il s'agit pour les problèmes d'optimisation d'identifier un objectif pour mesurer la qualité d'un choix donné (gain financier, coût, temps, consommation de ressource ... etc.) en déterminant les éléments agissant sur cet objectif (variable de décision) et la manière dont ils interagissent entre eux (contraintes de système), ce processus est appelé modélisation.

Bien que les problèmes d'optimisation combinatoire soient souvent faciles à définir et à modéliser, leurs résolutions est généralement difficile, et la plupart de ces problèmes appartiennent à la classe des problèmes NP-difficile et ne possèdent donc pas à ce jour des solutions algorithmiques efficaces pour toutes les données **[2].**

Dans ce chapitre, nous présentons d'abord les éléments de base de cette discipline et sa relation avec la théorie de la complexité, en suite nous donnerons une classification simple des diff érentes classes de problèmes d'optimisation, et nous terminerons par quelques exemples et certaines méthodes de résolution.

2. Problème d'optimisation combinatoire

2.1 Definition

L'optimisation combinatoire est minimiser ou maximiser une fonction souvent appelée fonction coût, d'une ou plusieurs variables soumises à des contraintes. L'optimisation combinatoire occupe une place très importante en recherche opérationnelle, en mathématiques discrètes et en informatique. Son importance se justifie d'une part par la grande difficulté des problèmes d'optimisation et d'autre part par de nombreuses applications pratiques pouvant être formulées sous la forme d'un problème d'optimisation combinatoire **[3]**.

D'un point de vue mathématique, un problème d'optimisation combinatoire est une minimisation ou bien une maximisation d'une certaine fonction sous des contraintes. Notons par :

•x : le vecteur constituer de variables de décision, avec $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$,

• f : la fonction objectif réelle,

•g : le vecteur des fonctions intervenant dans les contraintes.

Un problème d'optimisation peut alors être défini en général de manière suivante :

$$Minimiser\ f(X) \qquad (1.1)$$

$$\boldsymbol{S.C}\ \ g_i(X) = 0, l \in l_1$$

$$g_i(X) \leq 0, i \in l_2$$

Où I_1 et I_2sont les ensembles d'indices des contraintes de types égalité et inégalité respectivement.

2.2 Classification des problèmes d'optimisation

Face à la résolution d'un problème d'optimisation, il est important de bien identifier à quelle catégorie ce problème appartient. En eff et, les algorithmes développés sont conçus pour résoudre un type de problème donné et sont peu effi caces pour un type diff érent. La classification des problèmes d'optimisation change d'un auteur à l'autre. Par exemple, on distingue :

- **Les problèmes d'optimisation continue et les problèmes d'optimisation discrète**

Dans certains cas, les variables de décision sont discrètes, le plus souvent sous la forme d'entiers ou de binaires. Le problème d'optimisation est dit discret. Au contraire, dans les problèmes d'optimisation continue, les variables peuvent prendre n'importe quelle valeur, ce sont des réels. Les problèmes d'optimisation continue sont généralement plus simples à résoudre. Un problème d'optimisation mêlant variable continue et variable discrètes est dit mixte.

- **Les problèmes d'optimisation avec ou sans contrainte**

Il est important de bien distinguer les problèmes où des contraintes existent sur les variables de décision. Ces contraintes peuvent être simplement des bornes et aller jusqu'à un ensemble d'équations de type égalité et de type inégalité. Il est parfois possible d'éliminer une contrainte égalité par substitution dans la fonction objectif. Naturellement, les problèmes avec contraintes sont plus compliqués à résoudre et utilisent des algorithmes dédiés.

- **Les problèmes d'optimisation mono-objectif ou multi-objectif**

Les problèmes mono-objectif sont définis par une unique fonction objectif. Les problèmes multi-objectifs existent quand un compromis est à rechercher entre plusieurs objectifs contradictoires. Il est éventuellement possible (mais pas nécessairement efficace) de reformuler un problème multi-objectif avec une seule fonction objectif sous forme d'une combinaison des différents objectifs ou en transformant des objectifs sous forme de contraintes.

- **Les problèmes d'optimisation déterministe ou stochastique**

Les problèmes d'optimisation déterministe considèrent que les données sont connues parfaitement, alors que dans les problèmes d'optimisation stochastique, ce n'est pas le cas par exemple une approche stochastique peut être pertinente dans le cas où les variables d'un problème sont les ventes futures d'un produit. Dans ce cas, l'incertitude peut être introduite dans le modèle.

2.3 Résolution d'un problème d'optimisation combinatoire

Résoudre un problème d'optimisation combinatoire nécessite l'étude de trois points particuliers :

— La définition de l'ensemble des solutions réalisables,

— L'expression de l'objectif à optimiser,

— Le choix de la méthode d'optimisation à utiliser.

Les deux premiers points relèvent de la modélisation du problème, le troisième de sa résolution. Afin de définir l'ensemble des solutions réalisables, il est nécessaire d'exprimer l'ensemble des contraintes du problème **[2]**

2.4 Méthodes de résolution

Souvent les problèmes d'optimisation combinatoire sont des problèmes NP-Complet, ils sont traités de diverses manières, la première manière qui prévaut à toutes les autres est d'essayer de trouver une solution réalisable à ce problème, puis on doit essayer d'améliorer la qualité de ces solutions. Pour ce faire il existe divers algorithmes pour la résolution des problèmes combinatoires, ces algorithmes sont illustrés dans la figure suivante. En décrivant ci-après leurs principales caractéristiques.

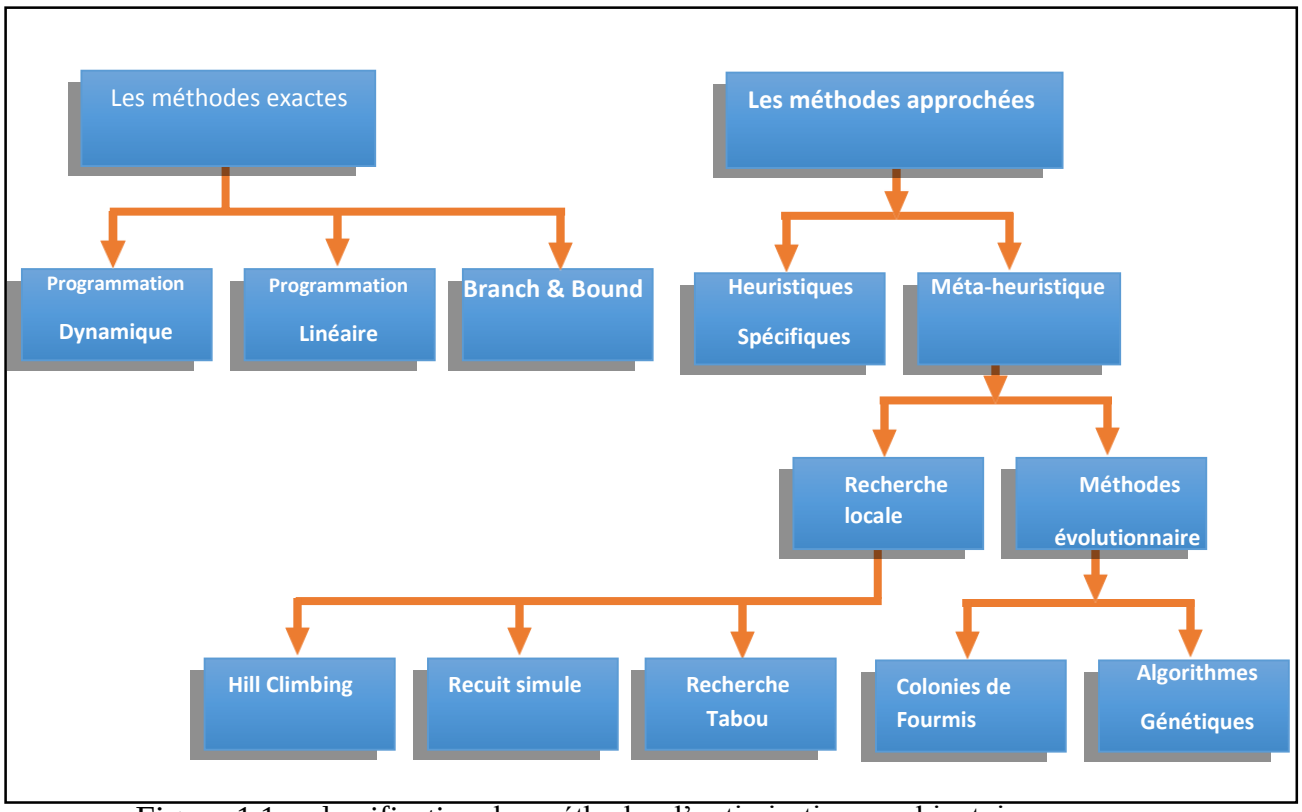

Figure 1.1 – classification des méthodes d'optimisation combinatoire

2.4.1 Méthodes exactes

Le principe des méthodes exacte consiste à chercher, la meilleure solution dans l'ensemble des solutions 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 classer les méthodes exactes en trois grandes classes suivantes :

— La programmation dynamiques **[4],[5].**

— La programmation linéaire et quadratique **[6],[2].**

— Les méthodes de recherche arborescente (branch and bound) **[7],[8].**

2.4.1.1 Programmation Dynamique

La programmation dynamique a été appelée comme cela depuis 1940 par Richard Bellman et permet d'appréhender un problème de façon différente de celle que l'on pourrait imaginer au premier abord. Le concept de base est simple : une solution optimale est la somme de sous-problèmes résolus de façon optimale. Il faut donc diviser un problème donné en sous-problèmes et les résoudre un par un.

- **Algorithme général de programmation dynamique**

La conception d'un algorithme de programmation dynamique peut être planifiée dans une séquence de quatre étapes.

1. Caractériser la structure d'une solution optimale.
2. Définir récursivement la valeur d'une solution optimale.
3. Calculer la valeur d'une solution optimale en remontant progressivement jusqu'à l'énoncé du problème initial.
4. Construire une solution optimale pour les informations calculées.

2.4.1.2 La méthode de branch and bound

La méthode de branch and bound (procédure par évaluation et séparation progressive) consiste à énumérer ces solutions d'une manière intelligente en ce sens que, en utilisant certaines propriétés du problème en question, cette technique arrive à éliminer des solutions partielles qui ne mènent pas à la solution que l'on recherche. De ce fait, on arrive souvent à obtenir la solution recherchée en des temps raisonnables. Bien entendu, dans le pire cas, on retombe toujours sur l'élimination explicite de toutes les solutions du problème.

Pour ce faire, cette méthode se dote d'une fonction qui permet de mettre une borne sur certaines solutions pour soit les exclure soit les maintenir comme des solutions potentielles.

Bien entendu, La performance d'une méthode de branch and bound dépend, entre autres, de la qualité de cette fonction (de sa capacité d'exclure des solutions partielles tôt).

Algorithme général

Début

Placer le nœud début de longueur *0* dans une liste.

Répéter

Si la première branche contient le nœud recherché **alors**

Fin avec succès.

Sinon

- Supprimer la branche de la liste et former des branches nouvelles en étendant la branche supprimée d'une étape.

- Calculer les coûts cumulés des branches et les ajouter dans la liste de telle sorte que la liste soit triée en ordre croissant.

Jusqu'à (liste vide ou nœud recherché trouvé)

Fin

2.4.2 Les méthodes approchées ou heuristique

2.4.2.1 Heuristiques

Définition

Une heuristique est une méthode généralement utilisée pour résoudre les problèmes de large étendue, non fondée sur un modèle formel et n'aboutit pas nécessairement à une solution optimale. Une méthode heuristique procède par évaluations successives et hypothèses provisoires. Elle constitue une alternative très intéressante pour traiter les problèmes d'optimisation de grande taille si l'optimalité n'est pas primordiale.

Types d'heuristiques

Généralement, une heuristique est conçue pour un problème particulier en s'appuyant sur sa structure propre sans offrir aucune garantit quant à la qualité de la solution calculée.

- **Heuristique constructive**

Ce sont des méthodes itératives qui construisent pas à pas une solution. Partant d'une solution partielle initialement vide, elles cherchent à étendre à chaque étape la solution partielle de l'étape précédente, et ce processus se répète jusqu'à ce que l'on obtienne une solution complète, la figue suivante montre le principe de l'approche constructive.

Ces méthodes sont généralement utilisables lorsque la qualité de solution n'est pas un facteur primordial ou la taille de l'instance est raisonnable, en l'occurrence pour générer une solution initiale dans une méta-heuristique, ces méthodes sont rapides et faciles à implémenter.

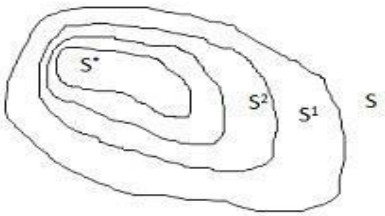

FIGURE 1.2 – Approche constructive

- **Heuristique de recherche locale**

Les méthodes de recherche locale, aussi appelées algorithmes de descente ou d'amélioration itérative, partent d'une solution initiale, et ont pour but de l'améliorer. voir figure la figure ci-dessous.

Le principe général de ces méthodes est le suivant : à partir d'une solution initiale x, dont on connaît la valeur de la fonction objective f (x), on cherche la meilleure solution x_0 dans le voisinage de x. Si l'on ne parvient pas à améliorer x, alors on s'arrête, sinon on remplace x par x_0, et on recommence.

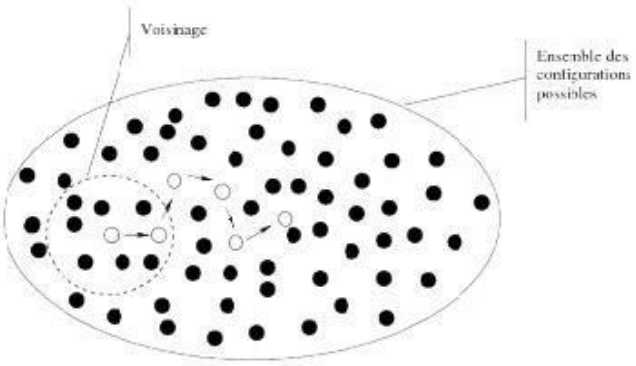

FIGURE 1.3 – Approche de recherche locale

2.4.2.2 Méta-heuristiques

Le mot méta heuristique est dérivé de la composition de deux mots grecs :

- heuristique qui vient du verbe heuriskein et qui signifie 'trouver'.

- méta qui est un suffixe signifiant 'au -delà', 'dans un niveau supérieur'.

Les méta-heuristiques sont des méthodes inspirées de la nature, ce sont des heuristiques modernes dédiées à la résolution des problèmes et plus particulièrement aux problèmes d'optimisation, qui visent à atteindre un optimum global généralement enfoui au milieu de nombreux optimums locaux.

Les méta- heuristiques se subdivisent en deux sous-classes :

Les méthodes de voisinage

Ces méthodes partent d'une solution initiale (obtenue de façon exacte, ou par tirage aléatoire) et s'en éloignent progressivement pour réaliser une trajectoire, un parcours progressif dans l'espace des solutions. Dans cette catégorie, se rangent :

— Le recuit simulé **[9].**

— La recherche Tabou **[10],[11].**

- **Le recuit simulé**

La méthode du recuit simulé est une généralisation de la méthode Monte-Carlo; son but est de trouver une solution optimale pour un problème donné. Elle a été mise au point par trois chercheurs de la société IBM : S. Kirkpatrick, C.D. Gelatt et M.P. Vecchi en 1983

- **La recherche Tabou**

La méthode de recherche tabou, ou simplement méthode tabou, a été formalisée en 1986 par F. Glover **[12]**. Sa principale particularité tient dans la mise en œuvre de mécanismes inspirés de la mémoire humaine. L'idée consiste à garder la trace du cheminement passé dans une mémoire et de s'y référer pour guider la recherche.

À chaque itération, l'algorithme tabou choisit le meilleur voisin non tabou, même si celui-ci dégrade la fonction de coût. Pour cette raison, on dit de la recherche avec tabou qu'elle est une méthode agressive.

Les méthodes évolutives

Ces méthodes se distinguent de celles déjà étudiées par le fait qu'elles opèrent sur une population de solution ; pour cela, elles sont souvent appelées des méthodes à base de population. Certaines d'entre elle ont des principes inspirés de la génétique et du comportement des insectes ; on trouve entre autre : les algorithmes génétiques, les colonies de fourmis **[13]**, les essaims particulières **[14],[15**]... etc. La complexité de ces phénomènes biologiques ont servi de modèle pour des algorithmes toujours plus sophistiqués ces vingt dernières années.

3. Eléments de la théorie de la complexité

La théorie de la complexité traite de la difficulté de résolution des problèmes formulés mathématiquement. Cette difficulté est mesurée ainsi bien en temps (temps nécessaire à sa résolution) qu'en espace (espace mémoire). Il est prévisible d'avoir une augmentation de ces quantités en fonction de la taille du problème, on s'intéresse alors à la manière dont elle augmente. La pire situation étant que le temps nécessaire à la résolution d'un problème soit une fonction exponentielle de la taille du problème. **[16]**

3.1 Notions de base

L'évaluation de la complexité en temps des algorithmes repose sur une définition précise du temps qui ne dépend pas de la vitesse d'exécution de la machine ni de la qualité du code produit par le compilateur. En effet, le temps de calcul obtenu par l'exécution d'un programme sur une machine donné ne reflète pas réellement la difficulté de celui-ci, donc ne devient pas plus facile lorsqu'un processus plus performant est sur le marché. Afin d'estimer plus finement la complexité d'un algorithme, il est nécessaire de s'abstraire de cette notion temporelle dépendante du matériel et de

considérer le temps comme le nombre d'opérations T P (n) nécessaire à l'exécution d'un programme pour une certaine donné de taille n.

Généralement, pour aborder l'étude des problèmes d'optimisation combinatoire il est nécessaire de se familiariser avec les termes suivants :

Définition (Problème de décision) [16]

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

Définition (Instance d'un problème) [16]

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 (Problème d'optimisation) [16]

Un problème d'optimisation est un problème caractérisé par un quadruplet hI ; S; f ; mode i 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 (min ou max) indique s'il s'agit de minimisation ou maximisation de la taille de ce résultat.

Définition (Réductibilité) [16]

Un problème X peut être ramené (réduit) à un autre problème X_0 si une instance quelconque de

X peut être facilement reformulée comme instance de X_0 dont la solution sera aussi solution pour X.

Définition (Réductibilité en temps polynomial) [16]

La fonction qui réduit le problème X en X_0 peut être obtenue en temps polynomial.

3.2 Classes de Complexité

La théorie de la complexité des algorithmes étudie formellement la différence intrinsèque des problèmes algorithmiques et il existe un très grand nombre de classes de complexité, elles ont été introduites pour classer les algorithmes selon leurs caractéristiques, on définit les trois grandes classes P, NP et NP-Complet.

La classe P

Il s'agit de la classe de complexité contenant les problèmes décisionnels pouvant être résolus par une machine de Turing déterministe pendant un temps de calcul polynomial.

La complexité en temps de calcul pour un problème X exprime un rapport entre la taille de ses données et le temps utilisé pour le résoudre, pour cette raison qu'on dit que ces problèmes appartenant à cette classe comme étant effecients ou traitables, bien qu'il soit naturel et raisonnable de considérer comme intraitables, dans le cas pratique, pour un problème qui demande un temps de l'ordre de grandeur aussi élevé. Cette classe est réputée contenir de nombreux problèmes naturels, comme le calcul du plus grand diviseur commun.

On pourrait alors généraliser cette classe avec l'inclusion de tout autre problème qui peut être résolu en temps polynomial pour une machine de Turing non déterministe. La classe qui en résulte prend le nom de NP et elle implique une relation triviale : P ⊆ N P.

La classe NP

La classe des problèmes NP regroupe l'ensemble de problèmes qui peuvent être résolus avec une machine de Turing non déterministe et admettent un algorithme polynomial pour tester la validité d'une solution du problème traité (i.e. il est possible de vérifier que la solution proposée est correcte en un temps polynomial par rapport à la taille du problème).

La Classe NP-Complet

Un problème de décision π est NP-Complet s'il satisfait les deux conditions suivante : π appartient à NP, et tout problème NP se réduit à π en temps polynomial **[17]**.La classe NP-complet contient un ensemble de problèmes les plus difficiles de NP, de tels problèmes apparaissent régulièrement. Citons entre autres, le problème du voyageur de commerce, celui du stable maximal, le calcul du nombre chromatique...etc. tous ces problèmes correspondent à une difficulté pratique pour trouver un algorithme efficace pour de très grands graphes (les seuls algorithmes déterministes connus étant de complexité non polynomiale, souvent exponentielle).

Cette difficulté n'empêche pas de mettre au point des heuristiques, c'est-à-dire qui n'assurent pas à coup sûr l'obtention d'une solution du problème, mais y parviennent assez souvent ou assez près. Après avoir utilisé une méthode heuristique, on peut conclure la recherche avec une méthode exhaustive, c'est-à-dire d'examen a priori des toutes les configurations plausibles.
La grande question ouverte dans ce domaine, et qui est classée par l'institut de mathématique Clay parmi les 7 problèmes du prix du millénaire, concerne l'inclusion des classes P et NP, la question est traditionnellement formulée ainsi : Est-ce-que N = N P ?

Cette question n'ait pas encore trouvé de réponse car de très nombreux problèmes fondamentaux s'avèrent être < NP -difficiles >, et aucun algorithme polynomial n'existe pour les résoudre, sauf si P=NP. Puisqu'il est tout de même important de pouvoir les résoudre, les scientifiques introduisent et utilisent diverses méthodes pour attaquer ces problèmes.

4 . Exemples de problème d'optimisation combinatoire

4.1 Problème du sac à dos

Le problème du sac à dos (en anglais Knapsack problem), modélise une situation analogue au remplissage d'un sac à dos, ne pouvant supporté plus d'un certain poids, avec tout ou une partie d'un ensemble donné d'objet ayant chacun un poids et une valeur. Les objets mis dans le sac à dos doivent maximiser la valeur totale, sans dépasser le poids maximum.

4.2 Problème de transport

Le problème du transport est un programme de minimisation de cout de transport. Il a pour but d'acheminer au moindre des marchandises depuis m origines vers n destinations. Autrement dit, le problème peut s'illustrer comme suit : partons des places A_i (dépôts, usines) ayant des offres des produit X_{ij} , et d'autres places B_j (points de vente,client), qui expriment des demandes sur les produits X_{ij},(X_{ij} représente la quantité des produits transportés de l'origine A_i vers la destination B_j) La formulation et la résolution d'un problème de transport a pour cible :

— La minimisation du cout de transport.

— L'obtention d'une demande excédentaire nulle c'est-à-dire satisfaire la demande dans toutes les places B_j et éliminer les fournitures dans toutes les place A_i.

5. Conclusion

Nous avons présenté dans ce chapitre quelques méthodes les plus connues pour la résolution des problèmes combinatoires. Ces méthodes sont généralement classées en deux catégories : les méthodes exactes et les méthodes approchées. Les méthodes exactes ont l'avantage de garantir l'obtention de la solution optimale. Cependant, elles présentent un inconvénient majeur qui celui du temps d'exécution important, car ces méthodes parcourent tout l'espace de recherche. Les méthodes approchées sont des algorithmes qui tendent à s'approcher de l'optimal en temps raisonnable. Ces algorithmes présentent l'avantage d'être rapides, mais ne garantissent pas l'obtention de la solution optimale. Néanmoins, ils sont perçus par un bon nombre de chercheurs comme étant un bon compromis entre le temps d'exécution et la qualité de la solution.

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 **[18].**

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.

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 $i\epsilon\{1 \dots n\}$, dispose d'un poids w_i de valeur entière et d'un profit u_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 :

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

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 :

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

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 :
$x_1 + x_2 \leq 6\ et\ 9x_1 + 5x_2 \leq 45$. Il est bien clair que les deux variables $x_1$et x_2 sont entières.
Le programme associé à ce problème est :

$$\begin{cases} \max z = 8x_1 + 5x_2 \\ s.c\ \ x_1 + x_2 \leq 6 \\ 9x_1 + 5x_2 \leq 45 \\ x_1 \text{et } x_2 \in \mathbb{N} \end{cases}$$

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 **([19]-[20])**. 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 :

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

1. **Programmation :** le terme de programmation faisant référence à l'idée d'organisation et de planification liée à la nature des phénomènes modélisés **[21].**

2. **« Résoudre »le problème** $(\boldsymbol{PL})$ **:** 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 × 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 :

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

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 $n \times m$.
– $c = (c_1, \ldots, c_n)'$: vecteur des coûts (profits), $c \in \mathbb{Z}^n$.
– $b = (b_1, \ldots, b_m)'$: vecteur colonne des seconds membres, $b \in \mathbb{Z}^m$.
– $x = (x_1, \ldots, x_n)'$: 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 :

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

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 :

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

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) :

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

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.

Conclusion

Nous avons présenté dans ce deuxième chapitre comment 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.

Chapitre 3

Méthodes exactes pour la résolution des problèmes de PLE

3.1 Introduction

Un programme linéaire a une interprétation géométrique simple. Les éléments de l'ensemble $S = \{x \in \mathbb{N}/Ax \leq b\}$ 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.

3.2 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.

3.2.1 Inégalités valides pour un programme linéaire

Définition 3.1 : [22] Une inégalité $\pi' x \leq \pi_0$ est une inégalité valide pour $X \in \mathbb{R}^n$ si $\pi' x \leq \pi_0 \ \forall x \in X$.

Proposition 3.1 : [22] Si $\pi' x \leq \pi_0$ est une inégalité valide pour$X = \{x : x \geq 0, Ax \leq b\} \neq \emptyset$, alors $\exists\, u > 0$ tels que $u'A \geq \pi'$ et $\pi' b \leq \pi_0$.

3.2.2 Inégalités valides pour un problème de PLE

Inégalités valides de Chvàtal-Gomory [23]

Soient $u \in \mathbb{R}_+^m$, $S = X \cap \mathbb{Z}^n$ où $X = \{x \in \mathbb{R}_+^m : Ax \leq b\}, A = (a_1, a_2, \dots a_n)$ 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* :

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

Car $x \in \mathbb{N}^n$, donc $\sum_{j=1}^{1} [u'\, a_j] x_j$ est entier.
Un résultat intéressant est que toute inégalité valide d'un problème de PLE peut être obtenue de cette façon.

Théorème 3.1 : [23] Si $\pi' x \leq \pi_0$ est une inégalité valide pour S avec μ 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 3.1

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* :

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

3.2.3 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 :

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

Considérons le programme linéaire relaxé de (PLE) :

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

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

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

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

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 $x_j + \sum_{j \in J_N} \bar{a}_{ij}\, x_j = \bar{b}_j \quad (i \in J_B)$, $\bar{A} = (\bar{a}_{ij}, i \in J_B, j \in J_N)$ 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 $0 \leq \{\bar{a}_{ij}\} < 1$.

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

$$\Leftrightarrow x_i + \sum_{j \in J_N} [\bar{a}_{ij}] x_j - [\bar{b}_i] = \{\bar{b}_i\} - \sum_{j \in J_N} \{\bar{a}_{ij}\} x_j, \qquad i \in J_B. \qquad (3.1)$$

Soit $\tilde{x}$ une solution réalisable de (PLR) à valeurs entières. L'égalité (3.1) vaut pour $\tilde{x}$, le côté gauche étant à valeurs entières. De plus,

$$\tilde{x}_i + \sum_{J \in J_N} [\bar{a}_{ij}] \tilde{x}_j \leq \tilde{x}_i + \sum_{j \in J_N} \bar{a}_{ij}\, \tilde{x}_j = \bar{b}_i, \qquad i \in J_B, \qquad (3.2)$$

comme la partie gauche de (3.2) est à valeurs entières, alors :

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

comme la formule (3.1) est valable pour toute solution réalisable, il s'ensuit que :

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

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

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

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 :

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

Aller à (4).

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

retourner à (2).

Exemple 3.2

Considérons le programme linéaire en nombres entiers suivant :

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

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 :

$$\frac{-8}{9}x_3 - \frac{2}{3}x_5 - \frac{2}{6}x_6 - \frac{1}{9}x_7 + x_8 = \frac{-5}{9}.$$

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 $z(x) = 32$.

3.3 Méthode de séparation et d'évaluation

3.3.1 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 **[24]**."Banch-and-*Bound*" est le terme très suggestif donné par Little et al. **[25]** 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 :

$$\max_{x \in S} \quad z(x), \tag{3.3}$$

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 [26]

On présente les sous-ensembles S_i que l'on considère pour des sommets *si* $(S_0 = S)$
Une séparation de S_i est une opération qui consiste à remplacer S_i par une collection S_i^* de sous-ensembles $S_{i_1}, S_{i_2}, \dots, S_{i_k}$ de S_i tels que $S_{i_1} \cup S_{i_2} \cup \dots \cup S_{i_k} = S_i$ (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 3.1 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 [26] [27]

L'exploration des noeuds 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 plu 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 :

$$f = \sum_{j=1}^{n} min\left(\{x_j\}, 1 - \{x_j\}\right).$$

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 $\bar{z}_j$ de la valeur optimale de z sur S_j en remplaçant le sous-problème par une relaxation

$$\max_{x \in T_j \supseteq S_j} z(x) \qquad \left(\bar{z}_j = -\infty, T_j = \emptyset\right). \tag{3.4}$$

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. $\bar{z}_j = \underline{z}_j$, 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. $\bar{z}_j \leq \underline{z}_0$, 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 $\underline{z}_0 = -\infty$).

3.3.2 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 noeud 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 noeuds à examiner explicitement est réduite **[28]**.

Posons

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

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

$$S_j = \left\{x \middle| A^j x = b^j x \geq 0, x \ entier\right\} \subseteq S. \tag{3.5}$$

Pour la relaxation, on utilise :

$$T_j = \left\{x \middle| A^j x = b^j, x \geq 0 \ \right\} \supseteq S_j. \tag{3.6}$$

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 :

$$S_j^* = \left\{S_j \cup \{x|\ x_{b_i} \leq \left[Y_{i_0}\right]\}, S_j \cup \{x|x_{b_i} \geq \left[Y_{i_0}\right] + 1\}\right\}. \tag{3.7}$$

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.

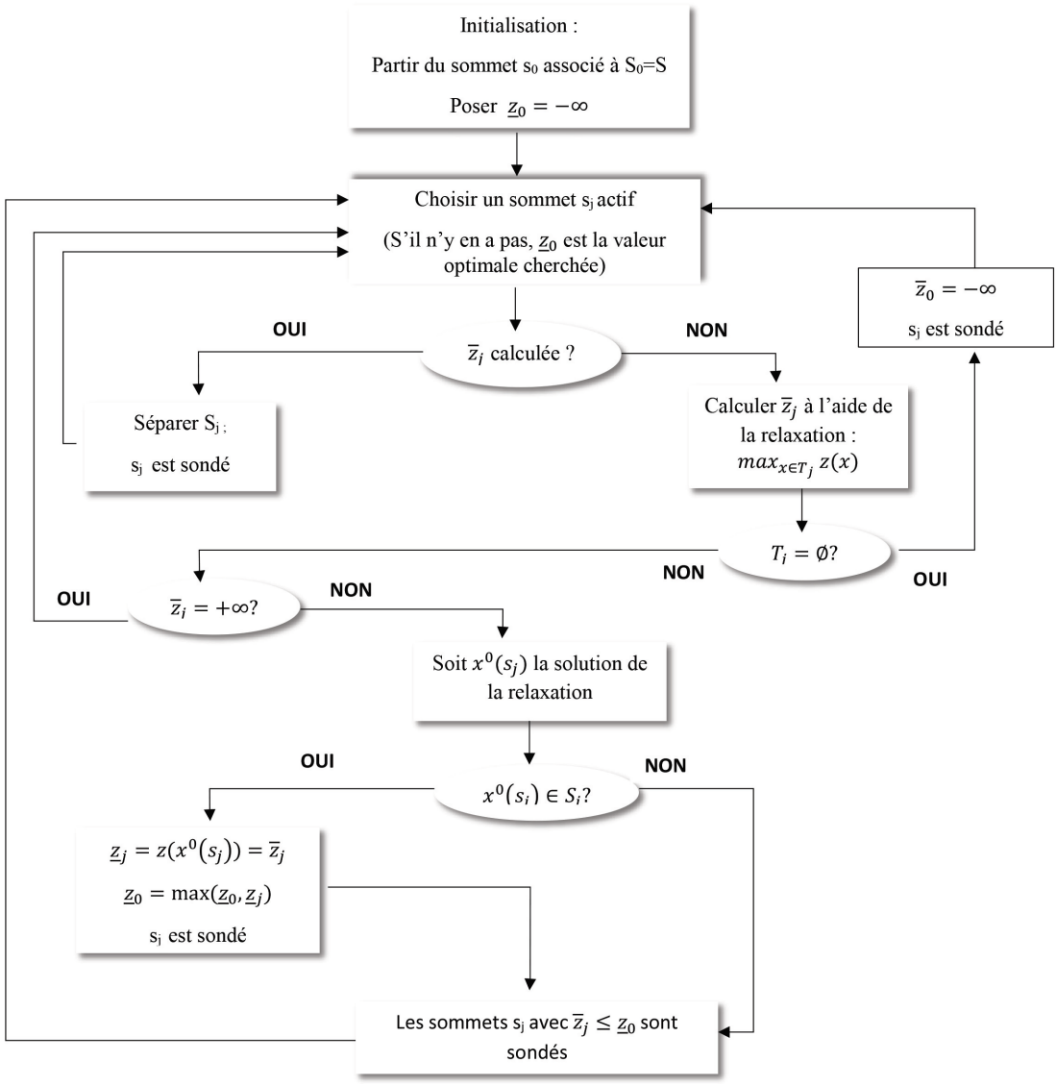

Figure 3.1 – Schéma de l'algorithme de *Branch-and-Bound*

Exemple 3.3

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

Soient $S_0 = S = \{x | Ax = b, x \geq 0\ entier$ et $\underline{z}_0 = -\infty$.
On va calculer $\underline{z}_0$ à l'aide de la relaxation :

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

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 :

$$S_0^* = \{\underbrace{S_0 \cup \{x|x_2 \geq 2}_{S_1}, \underbrace{S_1 \cup \{x|x_2 \geq 1\}}_{S_2}\}.$$
$$T_1 = \{x|Ax = b, x_2 \geq 2, x_1, x_2, x_3 \geq 0\}.$$

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 :

$$S_1^* = \{\underbrace{S_1 \cup \{x|x_1 \geq 7}_{S_3}, \underbrace{S_1 \cup \{x|x_1 \geq 6\}}_{S_4}\}.$$
$$T_{13} = \{x|Ax = b, x_2 \geq 2, x_1 \geq 7, x_1, x_2, x_3 \geq 0\}.$$

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

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

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 : $x = (2, 6, 0, 5)'$ qui elle est entière et $\bar{z}_4 = 32$, alors on coupe le sommetss_4.

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

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 :

$$S_2^* = \{\underbrace{S_2 \cup \{x|x_1 \geq 7}_{S_5}, \underbrace{S_2 \cup \{x|x_1 \leq 6\}}_{S_6}\}.$$
$$T_{25} = \{x|Ax = b, x_2 \leq 1, x_1 \geq 7, x_1, x_2, x_3 \geq 0\}.$$

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

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

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 : $x = (6, 1, 0, 6)'$ qui elle est entière et $\bar{z}_6 = 32$, alors on coupe le sommetss_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 $z = 32$.

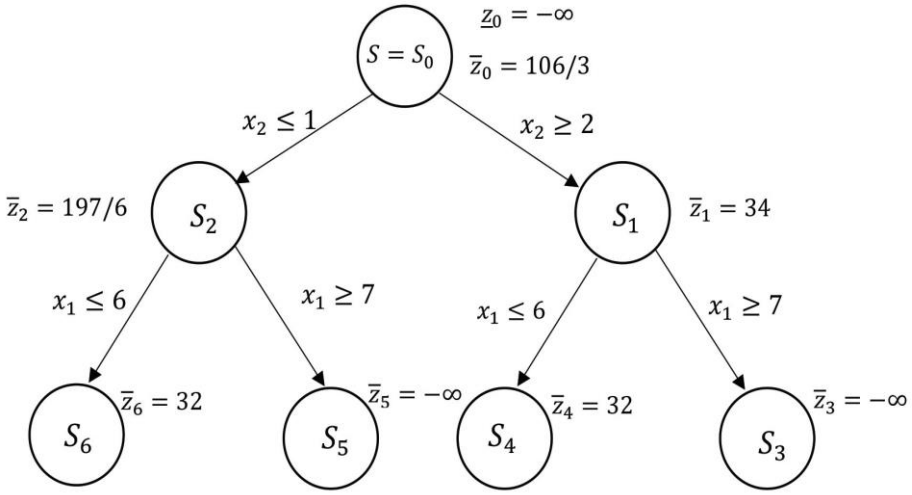

Figure 3.2 – Représentation sous forme d'arborescence

3.4 Programmation dynamique

3.4.1 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 :

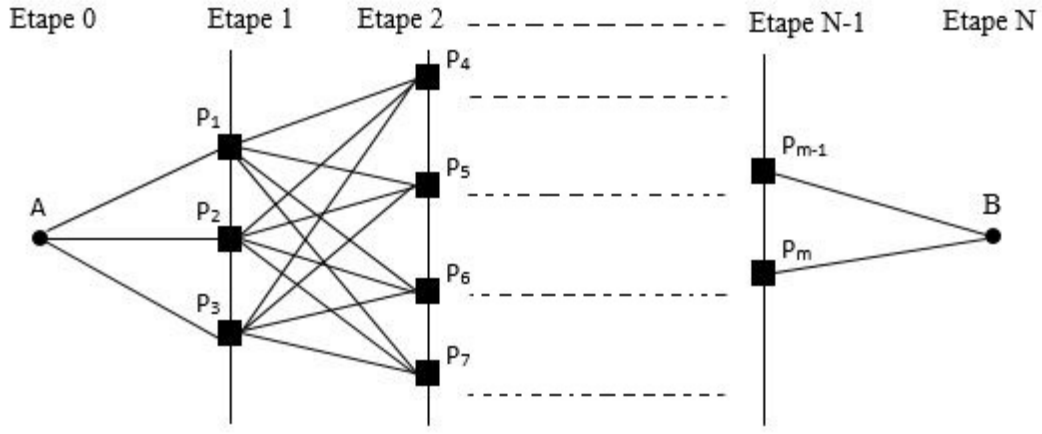

Figure 3.3 – Schéma représentatif de la programmation dynamique

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 $P_1, P_2, \dots, P_m$.

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.

3.4.2 Programmation dynamique pour la PLE

Principe de Bellman [29] : 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 :

$$\begin{cases} z = \max \sum_{j=1}^{n} c_j x_j \\ \sum_{j=1}^{n} c_j x_j \leq b \\ x_j \geq 0, \;\; x_j \; entier, \; j = 1, \dots n. \end{cases}$$

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

$$\beta_K(y) = \{\sum_{j=1}^{k} c_j x_j \mid \sum_{j=1}^{k} a_j x_j \leq y, x_j \geq 0, x_j \in \mathbb{N}, j = 1, \dots, k\} \; \dots (*)$$

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

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

Cette expression est équivalente à :

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

3.5 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.

3.5.1 Position du problème

On se basant sur les cours **[30]**, considérons le problème de PLE et à variables bornées, s'écrivant sous la forme canonique suivante :

$$(PLE) \quad \begin{cases} max \; z(x) = c'x \\ s.c \quad Ax = b \\ d^- \leq x \leq d^+ \;\; x \; entier, \end{cases} \qquad (3.8)$$

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 (3.8) suivant :

$$(PLR)\ \begin{cases} max\ z(x) = c'x \\ s.c \quad Ax = b \\ d^{-} \leq x \leq d^{+}. \end{cases} \tag{3.9}$$

Notons :
$I = \{1,2, \dots, m\}$: l'ensemble d'indices des lignes de A;
$J = \{1,2, \dots, n\}$: l'ensemble d'indices des colonnes de A.
Donnons les définitions suivantes :
– Un vecteur x vérifiant les contraintes $Ax = b, d^{-} \leq x \leq d^{+}, x$ entier, est appelé solution réalisable du problème (3.8). L'ensemble des solutions réalisables est alors donné par :

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

De même, on définit par $X = \{x \in \mathbb{R}^n : Ax = b, d^{-} \leq x \leq\ d^{+}\}$, l'ensemble des solutions réalisables du problème (3.9).
– Une solution réalisable x^0 est dite optimale pour le problème (2.9) si :

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

Où x^0 est une solution optimale du problème (3.9) et $\varepsilon \geq 0$ une précision choisie à l'avance.
– Soit un sous-ensemble d'indice $J_B \in J$ tel que $J = J_B \cup J_N,\ J = J_B \cap J_N = \emptyset, |J_B| = m.$
– En vertu de la partition de $J = J_B \cap J_N$, on peut alors écrire et fractionner les vecteurs et les matrices de la manière suivante :

$$x = \begin{pmatrix} x_B \\ x_N \end{pmatrix}, x_B = (x_j, j \in J_B), \qquad x_N = (x_j, j \in J_N);$$

$$c = \begin{pmatrix} c_B \\ c_N \end{pmatrix}, c_B = (c_j, j \in J_B), \qquad c_N = (c_j, j \in J_N);$$

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

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

L'ensemble J_B est alors appelé support si :

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

– 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 :

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

3.5.2 Formule d'accroissement de la fonction objectif

Soit (x, J_B) un plan de support du problème (3.9). On considère une autre solution réalisable quelconque $\bar{x} = x + \Delta x$. On a donc :

$$\Delta z = z(\bar{x}) - z(x) = c'\bar{x} - c'x = c'\Delta x. \quad (3.10)$$

Par ailleurs on a :

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

D'où :

$$\Delta x_B = -A_B^{-1}A_N\Delta x_N. \quad (3.11)$$

Ainsi, l'accroissement (3.10) devient :

$$\Delta z = c'\Delta x = -c'_B A_B^{-1}A_N\Delta x_N + c'_N\Delta x_N = -(c'_B A_B^{-1}A_N + c'_N)\Delta x_N. \quad (3.12)$$

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

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

Où $E' = (E'_B, E'_N)$, avec :

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

$$E'_N = u'A_N - c'_N = c'_N A_N^{-1}A_N - c'_N = 0,$$

Finalement, la formule d'accroissement (3.12) prend la forme finale suivante :

$$\Delta z = E'_N\Delta x_N = -\sum_{j\in J_N} E_j\left(\bar{x}_j - x_j\right). \quad (3.13)$$

3.5.3 Critère d'optimalité

On a le théorème d'optimalité suivant :

Théorème 3.2 Soit (x, J_B) un plan de support du problème Les relations :

$$\begin{cases} E_j \geq 0, \ pour \ x_j = d_j^-, \\ E_j \leq 0, \ pour \ x_j = d_j^+, \\ E_j = 0, \ pour \ d_j^- \leq x_j \leq d_j^+, \ j \in J_N, \end{cases} \quad (3.14)$$

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 (Condition suffisante) Soit un plan de support vérifiant les relations (3.14). Pour tout plan de support $\bar{x}$ du problème (3.9), la formule d'accroissement (3.13) donne

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

Où

$$d_j^- \le \bar{x}_j \le d_j^+ \implies \bar{x}_j - d_j^- \ge 0 \; et \; \bar{x}_j - d_j^+ \le 0,$$

Donc

$$z(\bar{x}) - z(x) \le 0 \Longrightarrow z(\bar{x}) \le z(x), \forall\, \bar{x}\; solution\; realisable.$$

Par conséquent, le vecteur x est une solution optimale du problème (3.9).

Condition Nécessaire

Soit $\{x, J_B\}$ un plan de support optimal non dégénéré du problème (3.9), et supposons que les relations d'optimalité (3.14) 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 :

$$\begin{cases} l_{j_0} = -sign \;\; E_{j_0}, \\ l_j = 0, \quad j \ne j_0 \quad j \in J_N, \\ l_B = l(J_B) = -A_B^{-1} A_N l_N = A_B^{-1} a_{j_0} sing \; E_{j_0}. \end{cases} \tag{3.15}$$

Le vecteur $\bar{x}$ vérifie les contraintes principales $A\bar{x} = b$. Pour que $\bar{x}$ soit une solution réalisable du problème (3.9), il doit aussi vérifier l'inégalité :

$$d^- \le \bar{x} \le d^+ \Leftrightarrow \; d^- \le x + \theta l \le d^+ \Leftrightarrow \; d^- - x \le \theta l \le d^+ - x.$$

C'est-à-dire :

$$\begin{cases} d_j^- - x_j \le \theta l_j \le d_j^+ - x_j, j \in J_B, \\ d_{j_0}^- - x_{j_0} \le -\theta sing \; E_{j_0} \le d_{j_0}^+ - x_{j_0}. \end{cases} \tag{3.16}$$

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

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

Par conséquent pour un nombre positif θ assez petit, le vecteur $\bar{x} = x + \theta l$ est une solution réalisable du problème (3.9) et l'accroissement (3.13) nous donne :

$$\Delta z = -\sum_{j\in J_N} E_j\left(\bar{x}_j - x_j\right) = -\theta \sum_{j\in j_n} E_j l_j = \theta E_{j_0} sing \; E_{j_0} = \theta |E_{j_0}| > 0, \tag{3.17}$$

c'est-à-dire $z(\bar{x}) - z(x) > 0 \Longrightarrow z(\bar{x}) > z(x)$, ce qui constitue une contradiction avec le fait que x est une solution optimale du problème (3.9).
Par conséquent, les relations (3.14) sont nécessairement vérifiées si $\{x, J_B\}$ est un plan optimal du problème (3.9).

3.5.4 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 (3.13) et on aura :

$$z(x^0) - z(x) = \sum_{j \in J_N} E_j(x_j^0 - x_j) = \sum_{E_j > 0, j \in J_N} E_j\,(x_j - x_j^0) + \sum_{E_j < 0, j \in J_N} E_j\,(x_j - x_j^0). \quad (3.18)$$

On a

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

D'où

$$x_j - x_j^0 \leq x_j - d_j^- \; et \; x_j - x_j^0 \geq x_j - d_j^+.$$

Donc

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

Ainsi, on obtient :

$$z(x^0) - z(x) = \sum_{E_j > 0, j \in J_N} E_j\left(x_j - d_j^-\right) + \sum_{E_j < 0, j \in J_N} E_j\left(x_j - d_j^+\right). \quad (3.19)$$

Le nombre

$$\beta(x, J_B) = \sum_{E_j > 0, j \in J_N} E_j\left(x_j - d_j^-\right) + \sum_{E_j < 0, j \in J_N} E_j\left(x_j - d_j^+\right), \quad (3.20)$$

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

Théorème 3.2 Soit $\{x, J_B\}$. un plan de support du problème (3.9) et $\varepsilon \geq 0. Si\ \beta(x, J_B) \leq \varepsilon$ alors x est ε-optimal.
Preuve En vertu de (3.20), on obtient :
$z(x^0) - z(x) \leq \beta(x, J_B) \leq \varepsilon \Longrightarrow$ x est ε-optimal.

3.5.5 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 (3.19) soit maximal, il faut prendre θ aussi grand que possible et choisir l'indice j_0 tel que :

$$\left|E_{j_0}\right| = \max_{j \in J_{N\,NO}} |E_j|,$$

Où $J_{N\,NO}$ représente le sous-ensemble des indices non basiques non optimaux, tel que :

$$J_{N\,NO} = \{j \in J_N : [E_j > 0 \quad \wedge \quad x_j > d_j^-] \vee [E_j < 0 \quad \wedge \quad x_j < d_j^+]\}.$$

D'autre part, le pas _ doit vérifier les relations (3.16) et la direction l est calculée suivant les relations (3.15).
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 (3.16), se calcule comme suit :

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

Pour l'indice j_0, on obtient :

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

Le pas maximal θ^0 est donc :

$$\theta^0 = \min\{\theta_{j_1}, \theta_{j_0}\},\ j_1 \in J_B, \quad j_0 \in J_N. \tag{3.21}$$

Donc la nouvelle solution réalisable est : $\bar{x} = x + \theta^0 l$.
Calculons l'estimation de suboptimalité pour le plan $\{\bar{x}, J_B\}$:

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

On a

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

En remplaçant, on aura :

$$\beta(\bar{x}, J_B) = \beta(x, J_B) - \theta^0 \left|E_{j_0}\right|. \tag{3.22}$$

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 $\theta^0 = \theta_{j_0}$ on pose :

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

3.6 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 (3.8), en se basant sur le travail **[27].**

Le principe est le même que celui des algorithmes de plans sécants. On commence par résoudre le problème relaxé (3.9) 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 (3.9),
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 $i \in J_B$:

$$Z_i(x) = -\left\{\tilde{b}_i - \sum_{j \in J_N^+ \cup J^+} x_{ij} d_j^- - \sum_{j \in J_N^- \cup J^-} x_{ij} d_j^+\right\} + \sum_{j \in J_N^+ \cup J^+} \{x_{ij}\}(x_j - d_j^-) + \sum_{j \in J_N^- \cup J^-} \{-x_{ij}\}(-x_j + d_j^+), \tag{3.23}$$

Où :
$\bar{b} = (A(I, J_B))^{-1} b = A_B^{-1} b$ et $X = \left(x_{ij}, i \in J_B, j \in J_N\right) = (A(I, J_B))^{-1} A_N = A_B^{-1} A_N$.

Proposition 3.2 $Z_i(x^e)$ est un nombre entier, quelque soit $\{x^e, J_B\}$ plan de support du problème (3.8) et de plus, on a : $Z_i(x^e) \geq 0, \forall\, i \in J_B$.

Preuve Soit x^e une solution réalisable du problème (3.8), c'est à dire :

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

On a :

$$\begin{aligned} Ax^e = b \Longrightarrow\ & A_B x_B^e + A_N x_N^e = b, \\ \Longrightarrow\ & x_B^e = A_B^{-1}(b - A_N x_N^e), \\ \Longrightarrow\ & x_B^e = A_B^{-1} b - A_B^{-1} A_N x_N^e = \tilde{b} - X x_N^e, \\ \Longrightarrow\ & x_i^e = \tilde{b}_i - \textstyle\sum_{j\in J_N} x_{ij}\, x_j^e,\ \ \forall\, i \in J_B, \\ \Longrightarrow\ & \tilde{b}_i = x_i^e + \textstyle\sum_{j\in J_N} x_{ij}\, x_j^e,\ \ \forall\, i \in J_B, \qquad (3.24) \end{aligned}$$

En remplaçant $\tilde{b}$ dans la formule («3.23), on aura pour $i \in J_B$:

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

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

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

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

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

$$+ \sum_{\substack{j\in J_N^-\cup J^- \\ \{x_{ij}\}\neq 0}} (d_j^+ - x_j^e)\,. \qquad (3,25)$$

On sait que :

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

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

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

entière de (3.8) .

D'autre part, on peut écrire

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

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

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

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

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

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

En remplaçant la formule (3.25), on aura :

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

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

Proposition 3.3 Soit $\{x^0,J_B^0\}$ un plan de support optimal de (3.9). S'il existe un indice $i\in J_B^0$ tel que $Z_{i_1}(x^0)<0,$ alors l'inégalité

$$Z_{i_1}\geq 0,$$

donne une coupe valide.

Preuve Évident, car on a démontré que pour une solution réalisable x^e du problème (3.8), on a $Z_{i_1}(x^0)\geq 0,\forall i\in J_B$.

Proposition 3.4 Soit $\{x^0,J_B^0\}$ un plan de support optimal du problème (3.9). 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 Si $\{x^0, J_B^0\}$ est un plan de support optimal basique du problème (3.9), 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 :

$$\begin{aligned} Ax^0 &= b \\ \Longrightarrow A_B x_B^0 + A_N x_N^0 &= b, \\ \Longrightarrow x_B^0 &= A_B^{-1} b - A_B^{-1} A_N x_N^0, \\ &= \tilde{b} - X x_N^0, \\ \Longrightarrow \tilde{b} &= x_B^0 + X x_N^0, \\ \Longrightarrow \tilde{b}_i = x_i^0 + \sum_{j \in J_N^+ \cup J^+} x_{ij} d_j^- &+ \sum_{j \in J_N^- \cup J^-} x_{ij} d_j^+, \quad i \in J_B^0. \end{aligned}$$

D'où

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

Proposition 3.5 Quand la solution réalisable x^0 est basique, donc on a :

$$Z_i(x^0) \leq 0, \qquad \forall i \in J_B.$$

Schéma de l'algorithme

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

Étape 1 poser $t = 0, I_0 = I = \{1,2, \dots, m\}$ et $J^0 = J = \{1,2, \dots, n\}$.
Soit $\{x^*, J_B^*\}$ un plan de support initial du problème (3.9).
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 (3.8).

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) $\exists t_1 \in J_B^0 : Z_{i_1}(x^0) < 0$. dans ce cas l'inégalité :

$$Z_{i_1}(x) \geq 0 \tag{3.26}$$

donne une coupe valide.
On incrémente t, $t = t + 1$.
On ajoute la contrainte (3.26) au système (3.9) et on résout le problème :

$$\begin{cases} max\ z(x) = c'x, \\ Ax = b, \\ Z_{i_1} \geq 0, \\ d^- \leq x \leq d^+. \end{cases} \tag{3.27}$$

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

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

où λ sera trouvé à partir de l'équation $Z_{i_1}(\bar{x}) = 0$.
– Comme on a ajouté une inégalité au système (3.27), 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 $I_t = T_{t-1} \cup \{m + t\}$ et $J^t = J^{t-1} \cup \{n + t\}$.
Soit $\{x^1, J_B^1\}$ le plan de support optimal de (3.27).
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 (3.4). 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 (3.27).

Preuve on a

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

où, $A\Delta x = 0$ vu que x^0 et x^* sont deux solutions réalisables de (3.9).
Le vecteur $\bar{x}$ vérifie la contrainte principale $A\bar{x} = 0$. 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 (3.27), 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 (3.27).
$\bar{J}_B$ est bien un support, puisque :

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

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

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

$$+ \sum_{j \in J_N^- \cup J^-} \{-x_{i_1 j}\}\,(-x_j^* - \lambda(x_j^0 - x_j^*) + d_j^+)) = 0,$$

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

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

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

d'où

$$\lambda = \frac{Z_{i_1}(x^*)}{\sum_{\substack{j \in J_N^- \cup J^- \\ \{x_{i_1 j}\} \neq 0}} (x_j^0 - x_j^*) - \sum_{j \in J_N} \{x_{i_1 j}\}\,(x_j^0 - x_j^*)}. \qquad (3.28)$$

On a aussi

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

$$- \sum_{j \in J_N^- \cup J^-} \{-x_{i_1 j}\}(-x_j^* + d^+),$$

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

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

La formule (3.28) est équivalente à

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

Exemple 3.4
Soit à résoudre le problème de PLE suivant par la méthode de support :

$$\begin{cases} max\ z(x) = 3x_1 + x_2 + 2x_3 + 3x_4, \\ -x_1 + 3x_2 + x_3 - 2x_4 + x_5 = 17, \\ 7x_1 + \quad +3x_3 + x_4 + x_6 = 23, \\ 0 \le x_i \le 12,\ \ x_i \text{ entier},\ \ i = \overline{1,6}\,. \end{cases} \tag{3.29}$$

Le problème relaxé associé est :

$$\begin{cases} max\ z(x) = 3x_1 + x_2 + 2x_3 + 3x_4, \\ -x_1 + 3x_2 + x_3 - 2x_4 + x_5 = 17, \\ 7x_1 + \quad +3x_3 + x_4 + x_6 = 23, \\ \quad 0 \le x_i \le 12,\ \ i = \overline{1,6}\,. \end{cases} \tag{3.30}$$

Étape 1 : On résout par la méthode directe de support le problème (3.30).
Soit $x^* = (0,0,5,0,12,8)'$ et $J_B^* = \{3,1\}$ un plan initial de support du problème (3.30) avec $z(x^*) = 10$.
Après 5 itérations, nous avons trouvé le plan de support optimal suivant :

$$x^0 = (0,12,11/3,12,4/3,0)' \quad J_B^0 = \{3,5\} \quad \text{avec} \quad z(x^0) = 166/3.$$

Étape 2 : Le plan x^0 est non entier, on rajoute alors au programme (3.30) une coupe.
Itération 1 Le vecteur des estimations vaut :

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

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

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

On se trouve dans le cas 1. Alors on rajoute aux contraintes (3.30) la contrainte $Z_{i1}(x) \geq 0$ tel que 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 :

$$\begin{cases} max\ z(x) = 3x_1 + x_2 + 2x_3 + 3x_4, \\ -x_1 + 3x_2 + x_3 - 2x_4 + x_5 = 17, \\ 7x_1 + \quad +3x_3 + x_4 - 2x_6 = 23, \\ x_1 - x_4 + x_6 - 3x_7 = -22, \\ 0 \leq x_i \leq 12, \ \ i = \overline{1.6}, \ \ x_7 \geq 0. \end{cases} \qquad (3.31)$$

On résout le programme (3.31) en commençant par une solution notée $\bar{x}$ calculée comme suit :

$$\bar{x} = x^* + \lambda(x^0 - x^*), \text{où} \ \ \lambda = \frac{15}{16},$$

donc

$$\bar{x} = (0, \frac{45}{4}, \frac{15}{4}, \frac{45}{4}, 2, \frac{1}{2}, 0)', \qquad et \quad \bar{J}_B = \{3,5,7\}.$$

Après 6 itérations, nous avons obtenu une solution optimale de support qui est :

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

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

3.7 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 4

Méthodes approchées pour la résolution des problèmes de PLE

4.1 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 **[31].**

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 **[32].**

4.2 Un aperçu sur les heuristiques

De nombreux chercheurs comme Ackoff **[33]**, Müller-Merbach **[34]** 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 :

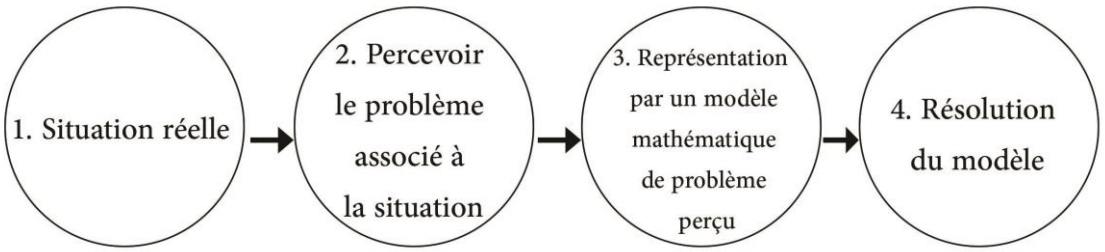

Figure 4.1 – D'une situation réelle à la solution du problème

4.2.1 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 **[32]** :

(i) Simples à mettre en oeuvre : Woolsey et Swanson **[35]** 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.

4.2.2 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.

4.2.3 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 **[36].**

4.3 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) **[37]** 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" **[20].**
Quand on utilise une heuristique pour pouvoir contrôler une autre, l'algorithme obtenu est connu sous le nom de métaheuristique.

4.3.1 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 **[38].**

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 le 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 **[39]** 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.

4.3.2 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 4.3.1 [40]

$$\begin{cases} \max z(x) = 20x_1+18x_2+15x_3 + 14x_4+12x_5 + 9x_6+7x_7 + 5x_8 + 3x_9+2x_{10} \\ s.c.\ 15x_1 + 16x_2 + 12x_3 + 12x_4 + 10x_5 + 10x_6 + 8x_7 + 5x_8 + 4x_9 + 3x_{10} \leq 45 \\ 22x_1 + 21x_2 + 16x_3 + 14x_4 + 15x_5 + 7x_6 + 5x_7 + 2x_8 + 4x_9 + 4x_{10} \leq 50 \\ 18x_1 + 20x_2 + 15x_3 + 10x_4 + 9x_5 + 8x_6 + 2x_7 + 6x_8 + 2x_9 + 5x_{10} \leq 40 \\ x_j \in \{0.1\} \quad j = 1 \dots 10, \end{cases}$$

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 :$\tilde{x} = (0.90,0,0,1,0,0.58,0.07,1,1,0,0)'$ 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 $x^2 = (0,0,0,1,0,0,1,10,0)'$, de valeur $z(x^2) = 26$, réalisable mais très éloignée de la valeur optimale.

4.3.3 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.

4.3.3.1 Le voisinage d'une solution

Définition 4.3.1 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 4.3.1 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) **[41]** ;

– Problème de "trim-loss" dans les rouleaux de papier (Ramirez-Beltran et Aquilar- Ruggiero, 1997) **[42]** ;

– Ordonnancement de capacité de production (Walser et al., 1998) **[43]** ;

– Ordonnancement de type jobshop avec des coûts retardés et avancés (Danna et al,.2003) **[44]**;

– Problème de tournée de véhicules avec fenêtre de temps (Danna, 2004) **[45]**.

4.3.4 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 **[47]** ;

– Algorithmes génétiques **[48]**.

4.3.5 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 ».

4.3.5.1 La génération de colonnes [49] [50]

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 sousproblème. 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 :

$$(PL)\begin{cases} max\ z = cx \\ sc \quad Ax = b \\ \quad x \geq 0. \end{cases}$$

On suppose que $n \gg m$ 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 $\mho$ des variables de (PL) et notons par A'la sous matrice de A correspondant à $\mho$ 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 $\bar{x} = B_K^{-1}b$;
   (b) Calculer les variables duales $c_{B_K}B_K^{-1}$;
   (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.

4.3.5.2 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.

4.3.6 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.

4.3.6.1 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.

4.3.6.2 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 $O(n\ long\ n + n \times f(n))$. Les algorithmes gloutons sont donc en général efficaces.

Exemple (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, $E = (e_i, \dots, e_n)$;
– 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 $S = (ne_i, \dots, ne_n$ 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**

$S[i] \leftarrow s\ div\ E[i]$;
$s \leftarrow s\ \ mod\ E[i]$;
$i++$;

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 (Sac-à-dos)

$$(P)\begin{cases} max\ z = 7x_1 + 3x_2 + 4x_3 + 3x_3, \\ sc\ \ 13x_1 + 8x_2 + 12x_3 + 10x_3 \leq 9, \\ x_j \in \{0,1\}\ \ J = 1, \dots, 4. \end{cases}$$

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	4/12	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 $x2$, 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 (Voyageur de commerce [23])
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 :

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

L'arc ayant une longueur minimale est $e_1 = (1,3)$ avec $c_{e1} = 2$. On sélectionne cet arc en posant $c_{e1} = 2$.
Le prochain arc avec une longueur minimale est $e_2 = (4,6)$ avec $c_{e2} = 3$ et $x_{e2} = 1$.
Posons $x_{e3} = 1$ tel que $e_3 = (3,6)$ et $c_{e3} = 6$.
Posons $x_{e4} = 4$ tel que $e_4 = (4,6)$ parce que le noeud 3 a été déjà visité, ainsi ces trois arcs : (1,3), (3,6), (2,3) ne peuvent pas être dans la tournée.
Posons $x_{e5} = 0$ tel que $e_5 = (1,4)$ car cet arc forme un cycle avec les arcs déjà choisis.
Posons $x_{e6} = 0$ tel que $e_6 = (1,2)$ avec $c_{e6} = 9$
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.

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.

Dans le travail que nous avons élaboré nous avons présenté les méthodes exactes et les méthodes approcher des problèmes de PLE.

Pour les méthodes exacte ou trouve la méthode gomory, la méthode de coupes, la méthode de dynamique, la méthode de séparation et d'évaluation et les méthodes direct du support. Ces méthodes nous donne une des solutions optimal à la fin mes dans un temps considérable.

Quant aux méthodes approchées nous avons présenté les heuristiques, le facteur temps pour ces méthodes est raisonnable mais les résultats sont pas vraiment optimale.

Néanmoins, existe d'autre méthodes qui sont plus puissant, à la fois en terme d'optimalité des résultats et dans un temps plut réduit qui sont l'hybridation des méthodes exacte et approcher

Bibliographie

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flows : theory, algorithms, and applications. 1993.

[2] M. J. Atallah and M. Blanton. Algorithms and theory of computation handbook, volume 2 : special topics and techniques. CRC press, 2009.

[3] A. H. Ben. Resolution du probleme de decoupe unidimensionnelle par une methode de generation de colonnes (french text). 1999.

[4] J. Desrosiers, Y. Dumas, and F. Soumis. A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. American Journal of Mathematical and Management Sciences, 6(3-4) :301–325, 1986.

[5] J. P. Heuristics. Intelligent search strategies for computer problem solving addison, 1984.

[6] C. Guéret, C. Prins, and M. Sevaux. Programmation linéaire. eyrolles. Technical report, ISBN 2-212-09202-4, 365 pages (in French), 2000.

[7] R. C. Corrêa, A. Ferreira, and S. C. Porto. Selected algorithmic techniques for parallel optimization. In Handbook of Combinatorial Optimization, pages 1879–1928. Springer, 1998.

[8] P. Schwerin and G. Wäscher. The bin-packing problem : A problem generator and some numerical experiments with ff d packing and mtp. International Transactions in Operational Research, 4(5-6) :377–389, 1997.

[9] P. Siarry. La méthode du recuit simulé : théorie et applications. Automatique-productique informatique industrielle, 29(4-5) :535–561, 1995.

[10] A. Hanset, H. Fei, O. Roux, D. Duvivier, and N. Meskens. Ordonnancement des interventions chirurgicales par une recherche tabou : Exécutions courtes vs longues. Logistique et Transport LTâĂŹ07, 2007.

[11] J. H. Holland. Genetic algorithms and the optimal allocation of trials. SIAM Journal on Computing, 2(2) :88–105, 1973.

[12] J. L. Bruno and P. J. Downey. Probabilistic bounds for dual bin-packing. Acta Informatica, 22(3) :333–345, 1985.

[13] M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. IEEE computational intelligence magazine, 1(4) :28–39, 2006.

[14] D. Datta and J. R. Figueira. A real-integer-discrete-coded particle swarm optimization for design problems. Applied Soft Computing, 11(4) :3625–3633, 2011.

[15] D. Liu, K. C. Tan, S. Huang, C. K. Goh, and W. K. Ho. On solving multiobjective bin packing problems using evolutionary particle swarm optimization. European Journal of Operational Research, 190(2) :357–382, 2008.

[16] T. Sofiane. Résolution de problemes de bin packing a une dimension par la programmation dc. Master's thesis, Université abederrahmane Mira Bejaia, 2013/2014.

[17] B. R. Tebbal mohamed. Etude de probléme de rangement ouvert avec conflits. Master's thesis, 2016.

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

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

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

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

[22] 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.

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

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

[25] 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.

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

[27] 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.

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

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

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

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

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

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

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

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

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

[37] 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.

[38] 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.

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

[40] 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.

[41] 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.

[42] 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.

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

[44] 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.

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

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

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

[49] 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

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

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