Université Cheikh r (UCAD)

Faculté Sciences et Techniques (FST)

Départements Mathématiques et Informatiques (DMI)

Mémoire Master II

Option : Marhématiques de la décision et Recherche opérationnelle

Titre :

Les Métaheuristiques en Optimisation

Par :

NGOM El hadji[1]

Jury :

Président	-	**Prof.**
Membres		**Prof.**
		Prof.
		Prof.
Superviseurs	-	**Prof. Salimata Guèye Diagne (UCAD)**
	-	**Prof. Youssou GNINGUE Université Laurentiane (Canada)**

Soutenue le .....

1. UCAD-AIMS-SENGAL

Dédicaces

Remerciements

Résumé

Abstract

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Contexte du travaill

Ce travail s'inscrit dans le domaine de résolution des problèmes d'optimisations difficiles avec l'utilisation des méthodes approximatives dites métaheuristiques. Dans ce type d'approches, on cherche à trouver des solutions pour des problémes classés "difficiles" en terme de temps de calcul pour les résoudre. Généralement, on peut dire qu'un problème est difficile si on ne peut pas localiser la meilleure solution de ce problème dans un temps raisonnable.

Deux approches sont généralement utilisées. La première cherche à trouver des solutions exactes, tandis que la seconde cherche à rapprocher le plus possible à la solution optimale, cette approche est concrétisée par des méthodes dites heuristiques, parmi elles, il existe des heuristiques applicables pour la majorité des problèmes d'optimisation, on les appelle les métaheuristiques.

Motivations

Plan du Travail

Chapitre Un

Les méthodes de résolution exactes et heuristiques

1.1 Introduction

Résoudre un problème d'optimisation linéaire en nombres entiers consiste à chercher la meilleure solution possible pour ce problème, définie comme la solution globabalement optimale ou un optimum global. La résolution de ces problèmes dits également combinatoires est souvent assez délicate puisque le nombre fini de solutions réalisables croît consiérablement avec la taille du problème, ainsi que sa complexité. Cette remarque a poussé les chercheurs à développer de nombreuses méthodes de résolution aussi bien qu'en Recherche opérqtioonelle (RO) qu'en intelligence artificielle (IA).
Dans la section suivante, nous allons d'abord définir les notions de complexité quant aux problèmes d'optmisation combinatoire.

1.2 Notions de compléxité

Avant de passer à un bref rappel sur les differntes méthodes de résolution exactes et heuristiques des problèmes d'optimisation linéaires en nombres entiers, nous introduisons quelques définitions et notions sur la complexité des ces derniers à savoir les PLNE.
La théorie de la complexité des algorithmes, née à la suite des travaux d'Emonds puis Cook et karp, a justement pour objet de lier le nombre de calculs éffectués lors de la résolution d'un problème au moyen d'un algorithme donné à la taille des données de ce problème. Nous ne ferons pas différence entre algorithme et programme.
En géneral, le temps d'éxcution est le facteur majeur qui détermine l'éfficacité d'un algorithme, alors la complexité en temps d'un algorithme est le nombre d'instructions nécessaires (affectation, comparaison, opérations algébriques , lecture et écriture, etc) que comprend cet algorithme pour une résolution d'un problème quelconque.

Définition 1.2.1. *Une fonction $f(n)$ est $O(g(n))$ ou ($f(n)$ est de complexité $g(n)$) s'il existe un réel $c > 0$ et un entier positif n_0 tels que pour tout $n > n_0$ on a $|f(n)| \leq c.g(n)$.*

Définition 1.2.2. *Un algorihme en temps polynomial est un algorithme dont le temps de la complexité est en $O(p(n))$, où p est une fonction polynomiale et n la taille de l'instance (ou sa longueur d'entrée).*
Si k est le dégré de ce polynome en n, le problème correspondant est dit être résoluble en $O(n^k)$ et appartient à la classe P. La connexité d'un grphe est un example de problème polynomial de classe P.

Définition 1.2.3. *La classe* ***NP*** *contient l'ensemble des problèmes de décision qui peuvent être décidés sur une machine non déterministe en temps polynomial. C'est la classe des problèmes qui admettent un algorihme en temps polynomial capable de tester la validité d'une solution du problème. Intuitivement, les problèmes de cette classe sont les problèmes qui peuvent être résolus en énumerant l'ensemble des solutions possibles et les tester à l'aide d'un algorihme polynomial.*

Définition 1.2.4. *On dit qu'un problème de recherche P_1 se réduit polynomialement à un problème de recherche P_2 par réduction de Turing s'il éxiste un algorihme A_1 pour résoudre P_1 en utilisant comme sous programme un algorithme A_2 résolvant P_2, de telle sorte que la complexité A_1 est polynomiale, quand on évalue chaque appel de A_2 par une constante.*

Définition 1.2.5 (La classe **NP-Complet**)**.** *Parmi l'ensemble des porblème des problèmes appartenant à* ***NP****, il éxiste un sous ensemble qui contient les problèmes les plus difficiles : on les appelle les problèmes* ***NP-Complets****. Un problème de classe* ***NP-Complet*** *possède la propriété que tous les problèmes* ***NP*** *lui sont réductibles. Si on trouve un algorithme polynomial à un problème* ***NP-Complet****, on trouve alors automatiquement une résolution polynomiale de tous les problèmes* ***NP*** *de la classe* ***NP****.*

Définition 1.2.6 (La classe **NP-difficile**)**.** *Un problème est* ***NP-difficile*** *s'il est plus difficile qu'un problème* ***NP-Complet****, c'est-à-dire s'il éxiste un problème* ***NP-Complet*** *se réduisant à ce problème par une réduction de Turing.*

Nous allons passer à la description des principales méthodes de résolution exactes des problèmes d'optimisation combinatoire.

1.3 Les méthodes exactes

Nous allons présenter d'abord quelconques méthodes de la classe des algorithmes complets ou exactes, ces méthdes donnent une garantie de trouver la solution optimale pour une instance de taille finie dans un temps limité et de prouver son optimalité [6].

1.3.1 La méthode du Simplexe

Parmi ces méthodes, on peut remarquer l'algorihme du simplexe qui permet d'obtenir la solution optimale d'un probléme d'optimisation en parcourant la fermeture convexe de l'ensemble de recherche. Son avantage c'est qu'il permet de résoudre un grand nombre de problèmes rapidement.
Voici la définition donnée par **Wikipédia, (consulté le 25/12/2015)** :

"*L'algorithme du simplexe est un algorithme de résolution des problèmes d'optimisation linéaire. Il a été introduit par George Dantzig à partir de 1947. C'est probablement le premier algorithme permettant de minimiser une fonction sur un ensemble défini par des inégalités. De ce fait, il a beaucoup contribué au démarrage de l'optimisation numérique. L'algorithme du simplexe a longtemps été la méthode la plus utilisée pour résoudre les problémes d'optimisation linéaire. Depuis les années 1985-90, il est concurrencé par les méthodes de points intérieurs, mais garde une place de choix dans certaines circonstances (en particulier si l'on a une idée des contraintes d'inégalité actives en la solution)*".

Cependant, l'algorihme du simplexe qui est bien entendu valable pour tout problème, il n'est nécessairement le plus efficace pour traiter des problèmes dont l'ensemble des contraintes présente certaines particularités (comme l'intégrité de la solution).

1.3.2 La Méthode de Séparation et d'Evauation

L'algorihme de séparation et d'évaluation, connu sous l'appellation anglaise **Branch and Bound (B&B)**, est l'une des methodes les plus connues pour résolution de problèmes d'optimisation combinatoires NP-Complets comme **le problème de sac à dos** [1]. Elle repose sur une méthode arborescente de recherche d'une solution optimale par séparations et évaluations, représentant les états solutions par un arbre d'états, avec des noeuds, et des feuilles.
Le branch and bound est basé sur tois axes principaux :

✎ **l'évaluation**,

✎ **la séparation**,

✎ **la stratégie de parcours**.

☞ **L'évalution** :

l'évaluation permet de rediure l'espace en éliminant quelques sous ensembles qui ne contiennent pas la solution optimale. L'objectif est d'essayer d'évaluer l'intérêt de l'exploration d'un sous ensemble de l'arborescence. Le branch and bround utilise une élimination de branches dans l' arborescence de recherche de la manière suivante : la recherche d'une solution de coût minimal, consiste à mémoriser la solution la moins coûteuse rencontrée pendant l'exploration, et à comparer le coût de chaque noeud parcouru à celui de la meilleure solution. Si le coût du noeud considéré est supéreiur au meilleur coût, on arrêt l'exploration de la branche et toutes les solutions de cette branche seront nécssairement de couût plus élevé que la solution déja troucée.

☞ **La séparation** :

la séparation consiste à diviser le problème en sous problèmes et en gardant la meilleure solution trouvée, on est assuré d'avoir résolu le problème inintial. cela revient à construire un arbre permettant d'énumerer toutes les solutions. L'ensemble des noeuds de l'arbre qu'il reste encore parcourir comme étant susceptibles de contenir une solution optimale, c'esr-à-dire encore à divser, est appelé ensemble des noeuds actifs. La procédure de séparation s'arrête losrqu'une des conditions suivantes est vérifée :

✓ on recoonait la meilleure solution de l'ensemble,

✓ on reconnait une solution meilleure que toutes celles de l'ensemble,

✓ on sait que l'ensemble ne cotient aucune soolution admissible.

☞ **La stratégie de parcours** :

- **La largeur d'abord :** cette stratégie favorise les sommets les plus proches de la racine en faisant moins de séparations du problème initial. Elle est moins efficace que les deux qui suivent,
- **La profonduer d'adord :** cette stratégie avantage les sommets les plus éloignés de la racine (de profondeur plus élevée) en applicant plus de séparations au problème initial. Cette voie mène rapidement à une solution optimale en économisant la mémoire,
- **La meilleur d'abord :** cette stratégie consiste à exploere les sous-problèmes possédant la meilleure borne. Elle permet aussi d'éviter l'exploration de tous les sous-problèmes qui possèdent une mauvaise évaluation par rapport à la valeur optimale. Elle dirige également la recherche là où la probabilité de trouver une meilleure solution est plus forte.

1. Knapsack Problem (KP)ou problème de sac à dos est un problème simple et classique d'optimisation combinatoire appartenant à la classe NP-Complet.

Représentation graphique de la décomposition par B&B appliquée au KP :
Formulation mathématique du KP : Dans ce cas, nous avons un sac à dos de maximal P et n oobjets. Pour chaque objet i, nous avons le poids p_i et une valeur ou utilité c_i.
variables de décision :

$$x_i = \begin{cases} 1 \text{ si l'objet i est mis dans le sac,} \\ 0 \text{ si l'objet i n'es pas mis dans le sac.} \end{cases}$$

On obtient modéle suivant :

$$(KP) \begin{cases} \max \sum_{i=1}^{n} c_i x_i \\ s.c \\ \sum_{i=1}^{n} p_i x_i \leq P \\ x_i \in \{0,1\} \end{cases}$$

La recherche par décomposition de l'ensemble des solutions peut être représentée graphiquement par un arbre (voir figure 1.1).
C'est de cette représentation que vient vient le nom de "méthode de recherche arborescente".

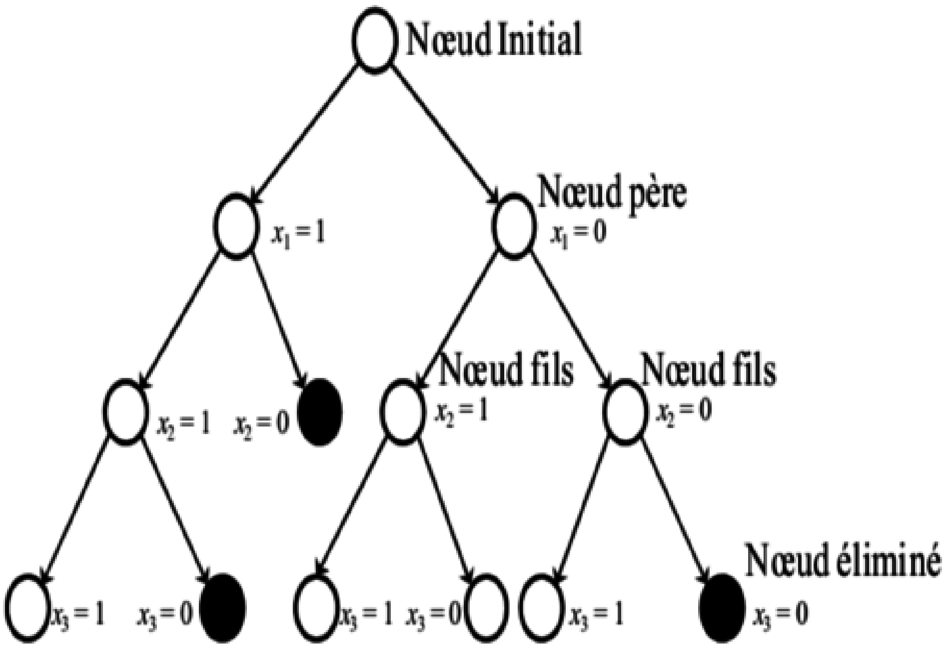


FIGURE 1.1 – Arbre engendré par décomposition d'un problème

1.3.3 Methode des coupes planes ou cutting planes méthod

Léthode des coupes à été développée par A. Schrijver dans son livre intitulé Theory of linear and integer programming en 1986 mais les travaux Ralph Edward Gomory ont rendu les cpues plus connues et plus efficaces. Elle est destinée à résoudre des problèmes d'optimisation linéaires en nombres entiers qui se formulent sous la forme standard d'un progrmamme linéaire (PL) :

$$(PL)\begin{cases}\min C^T x\\ s.c.\\ Ax \geq b\\ x \in \mathbb{R}^n\end{cases} \tag{1.1}$$

L'algorithme de cutting-plane forme une des classes des Algorithmes pour résoudre des PLE's qui utilise une idée sophistiquée trés intéressante. A chaque étape, on diminue la région réalisable en utilisant un plan coupant (cutting plane) jusqu'à ce que la solution optimale du PL (**relaxation** du PLE) soit entière correspondant la solution optimale du PLE inital. **Idée de base :**
On ajoute des contraintes linéaires au PLE qui n'excluent pas la solution entière réalisable.
Stratégie :

1. On ajoute des contraintes linéaires au PLE et donc auussi à la relaxation, une à chaque étape, jusqu'à la soluton optimale entière de la relaxation.
2. Puis qu'aucune solution réalisable du PLE n'est perdue par les coupes, alors la solution optimale entière de la relaxation du PLE ayant des contrintes ajoutées correspondra à la solution du PLE d'origine.

Observation :
Considérons un progrmamme en nombres entiers sous la forme standard

$$PLE(I)\begin{cases}\min c^T x\\ s.c.\\ Ax = b\\ x \in \mathbb{Z}^n_+\end{cases} \tag{1.2}$$

En supprimant la contrainte d'intégrité du PLE, on obtient le PL dit relaxation du PLE inital

$$PL(II)\begin{cases}\min c^T x\\ s.c.\\ Ax = b\\ x \in \mathbb{R}^n_+\end{cases} \tag{1.3}$$

On résout le PL obtenu par la méthode du simplexe (voir section 1.3.1) pour aboutir une solution optimale x^*.
Si tous les x_i^*, avec $i \in \{1, 2, \ldots, n\}$, sont entières, alors x^* correspond à x' la solution optimale du PLE, sinon on ajoute encore des contraintes puis on continue le processus ci-haut.

Remarque 1.3.1. *La méthode des coupes de Gomory nous permet de résoudre un PLE par usage de la méthode du simplexe, mais son principal inconvénient est que pour des problèmes raisonnables, l'algorithme peut converger parfois d'une manière trop lente vers la solution optimale.*

1.3.4 La méthode de Branch & Cut

La méthode **Branch & Cut** est une combinaison de la méthode de Branch and Bound et de la méthode de coupes de Gomory. Cette méthode améliore l'inefficacité et le manque de performance de ces deux méthodes face à certains problèmes appartenant à la classe

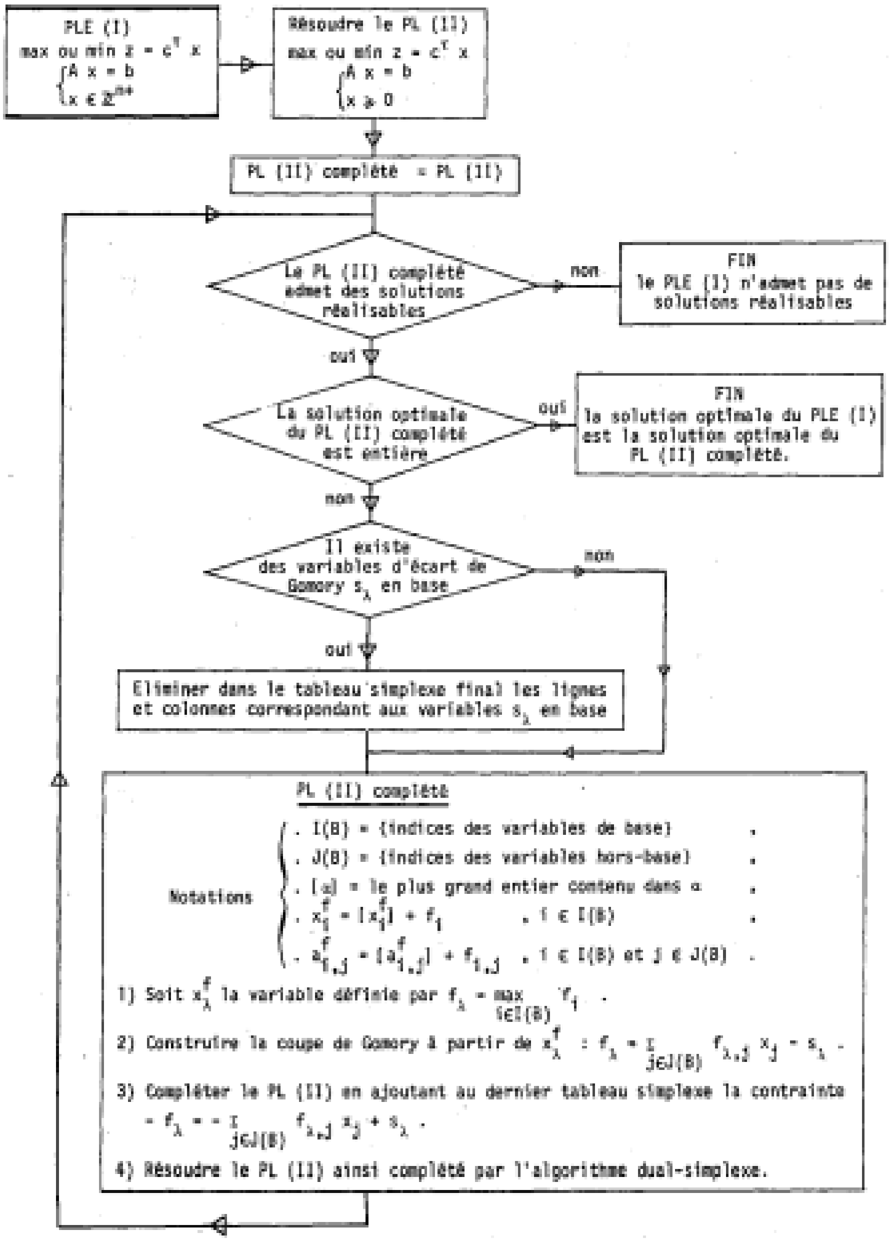


FIGURE 1.2 – Structure de la methode de Gomory, (cf support de cours : PE, p.164)

NP-difficile. Pour résoudre un PLE, la méthode de **Branch & Cut** commence d'abord par relaxer le problème puis appliquer les coupes planes sur la solution trouvée. Si on obtient une solution non entière, le problème sera divisé en sous-problèmes qui seront résolus de la même manière.
On se propose de résoudre le problème d'optimisation suivant par l'algorihme [1] :

$$(\min c^T x \ : Ax \geq b, \, x \in \mathbb{R}^n)$$

où $A \in \mathbb{R}^{m \times n}$ et $b \in \mathbb{R}^n$.

Algorithme 1 Branch and Cut

Liste des problèmes = *vide*
Initialser : le programme linèaire par le sous problème de contraintes
(A_1, b_1) avec $A_1 \in \mathbb{R}^{m_1 \times n}$ et $b_1 \in \mathbb{R}^{m_1}$ avec $m_1 << m$;
Etapes d'évaluation d'un sous problème :
Calculer la solution optimale $\bar{x}$ du programme linéaire $c^T \bar{x} = \min(c^T x \ : A_1 x \geq b_1, \, x \in \mathbb{R}^n)$;
Solution courante = Appliquer la méthode des coupes planes() ;
Fin étapes d'évaluation.
Si la solution courante est réalisable **Alors**
$x^* = \bar{x}$ est la solution optimale de $min(c^T x \ : Ax \geq b, \, x \in \mathbb{R}^n)$;
Sinon
Ajouter le problème dans Liste des sous problèmes ;
Fin du Si
Tant Que Liste des sous problèmes $\neq$ *vide* **Faire**
Sélectionner un sous problème ;
Brancher le problème ; Appliquer les étapes d'évaluation
Fin du Tant Que

Simulation numérique de [1]

L'algorihme de **Branch and Cut** est utilisé dans base du logiciel **IBM(R) ILOG(R) CPLEX(R) optimizer version 12.5.0.0** comme on va le voir sur le travail pratique suivant :

$$(PL) \begin{cases} \max(20x_1 + 8x_2 + 6x_3 + 5x_4 + 4x_5 x_6) \\ s.c. \\ 9x_1 + 8x_2 + 6x_3 + 5x_4 + 4x_5 x_6 \leq 12 \\ x_i \in \{0, 1\} \text{ avec } i \in \{1, 2, 3, 4, 5, 6\} \end{cases}$$

On écrit le PL sous l'extension **.lp** voir (1.4). On passe à la résolution du PL par le Cplex puis on ontient le résultat suivant sur son terminal :

```
\Problem name: exemple3.lp

Maximize
20 x1 + 8 x2 +6 x3 + 5 x4 + 4 x5 + x6
 Subject To
 c1: 9x1 + 8x2 + 6x3 + 5 x4 + 4x5 + x6 <= 12
BINARY
x1
x2
x3
x4
x5
x6
End
```

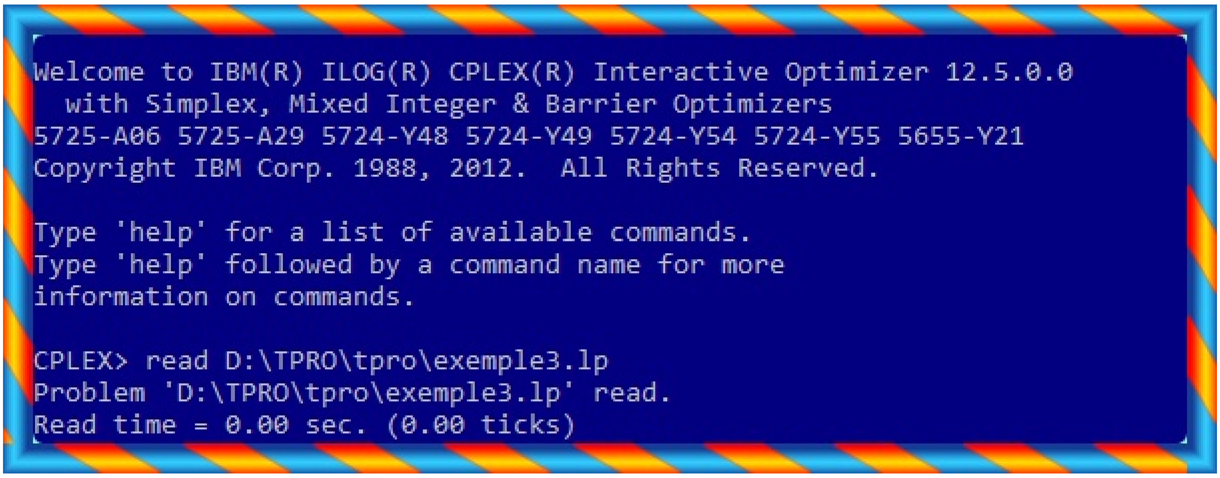
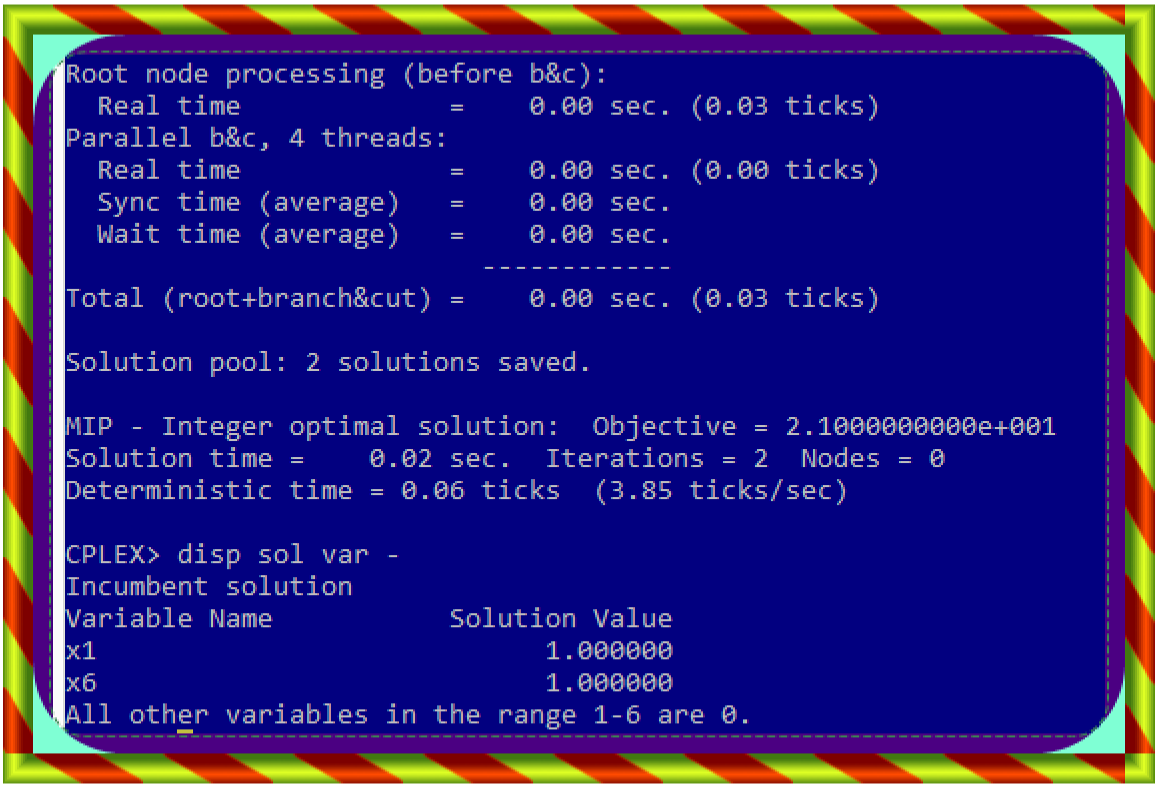


FIGURE 1.3 – Leture du PL au format “.lp” par Cplex.



```
Root node processing (before b&c):
  Real time             =    0.00 sec. (0.03 ticks)
Parallel b&c, 4 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    0.00 sec. (0.03 ticks)

Solution pool: 2 solutions saved.

MIP - Integer optimal solution:  Objective = 2.1000000000e+001
Solution time =    0.02 sec.  Iterations = 2  Nodes = 0
Deterministic time = 0.06 ticks  (3.85 ticks/sec)

CPLEX> disp sol var -
Incumbent solution
Variable Name           Solution Value
x1                            1.000000
x6                            1.000000
All other variables in the range 1-6 are 0.
```

FIGURE 1.4 – Details de résolution et affichage du résultat.

Remarque 1.3.2. *Il éxiste bien d'autres méthodes de résolution éxactes des problèmes d'optimisation comme la Programmation dynamique, la méthde de résolution par Colonies etc. A ce groupe s'ajoute les methodes heuristiques que nous allons voir en détails dans la section suivante.*

1.4 Les heuristiques

En Optimisation combinatoire, une **heuristique** est un algorihme d'approximaton qui permet d'identifier en temps polynomial au moins une solution réalisable rapidement, mais obligatoirement optimale. L'usage d'une heuristique est efficace pour Calculer une solution approchée d'un problème et ainsi accélérer le processus de résolution éxacte. Généralement une heuristique est conçue pour un problème particulier en s'appuyant sur sa structure propre sans offrir aucune garantie quant à la qualité de la solution Calculée. On distingue deux catégories d'heuristiques :

- Méthodes constuctives qui génèrent des solutions à partir d'une solution initiale en essayant d'en ajouter petit à petit des éléments jusqu'à ce qu'une solution complète soit obtenue.
- Méthodes de fouilles locales qui démarrent avec solution initialement complète (probablement moins intéressante) et de manière répétitive essaient d'améliorer cette solution en explorant son voisinage.

1.4.1 La descente recursive (recherche locale)

La méthode de descente est l'une des heuristiques classiques les plus connues. C'est un exemple typique de recherche locale, elle progresse à travers l'ensemble des solutions X par le choix de la meilleure solution voisinage de la courante et ainsi de suite ; ce processus s'interrompt dés que le premier minimum local est atteint. Pour un problème de minimisation d'une fonction f, l'algorithme de la descente peut être décrit comme suit :

Algorithme 2 DESCENTE RECURSIVE

Solution inintial s,
Répéter :
Si $f(s') < f(s)$ **Alors** $s := s'$,
Jusqu'à ce que $f(s') \geq f(s)$, $\forall s' \in N(s)$.

Remarque 1.4.1. *Cette heuristique est caractérisée par sa simplicité mais présente deux inconvénients :*

- *Suivant la taille et la structure du voisinage $N(s)$ considéré, la recherche de la meilleure solution voisine qui peut être aussi difficile que le problème initial,*
- *Elle est incapable de progresser au delà du premier minimum local rencontré. Par contre les problèmes d'optimisation combinatoires comportent en générale plusieurs optima locaux pour lesquels la valeur de la fonction objectif peut être fort éloignée 1.5,*
- *La descente recursive est la méthode de recherche locale la plus élémentaire donc elle est de trajectoire.*

1.5 Conclusion

Nous avons vu que comme les méthodes éxactes, les heuristiques classiques ne sont pas trés satisfaisantes pour résoudre efficacement en temps polynomial les problèmes d'optimisation présentant une certaine complexité. Les solutions issues de ces méthodes ne garantisent point la qualité de la solution Calculée. Ains les chercheurs s'inpirent de la nature pour créer de nouvelles méthodes plus générales et plus efficaces comme les métaheuristiques.

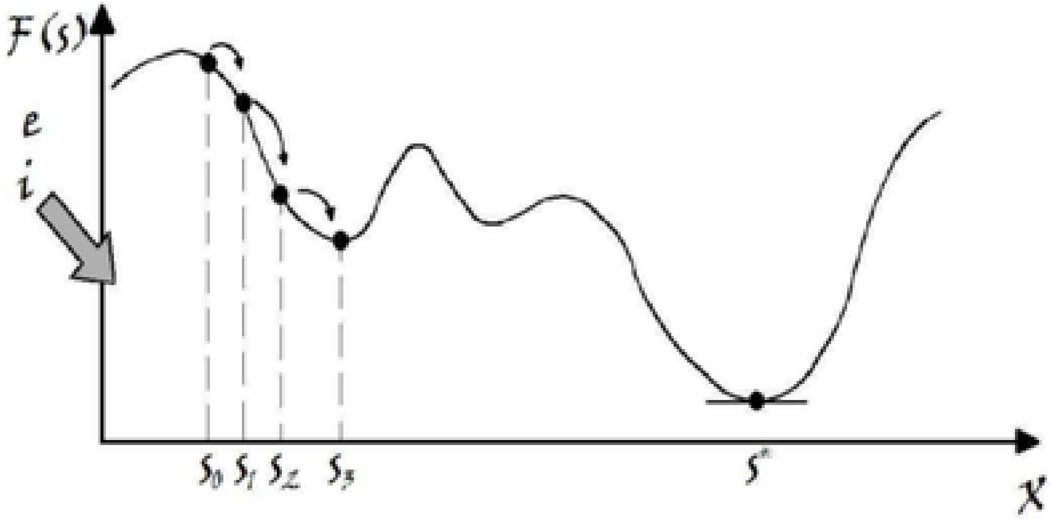


Figure 1.5 – Blocage de la descente recursive par un minimum local.

Chapitre Deux

Les métaheuristiques

2.1 Introduction

Définitions

Classification

2.2 Méthodes à base d'une seule solution

2.2.1 Le Recuit simulé

2.2.2 La recherche Tabou

2.3 Methodes à base de populations de solutions

2.3.1 Les colonies des fourmis

2.3.2 Les algo-génétiques

2.3.3 L'Optimisation par essaim de particule

2.3.4 La recherche dispersée

2.4 Conclusion

Chapitre Trois

Exemples pratiques d'applications

3.1 Introduction

3.2 Example 1

3.3 Example 2

3.4 Example 3

3.5 Conclusion

Conclusion générale

Bibliographie

[1] Acher J. ; Gadelle J : *Programmation Linéare.* Dunod Décision, Bordas, Paris, 1978, Troisième édition .

[2] Céa J. : *Optimisation théorique et Algorithmes.* Dunod, Paris, 1971.

[3] Minoux M. : *Programmation linéaire, théorie et algorithmes.* Dunond, Tomes 1 et 2

[4] Werra D. : *Eléments de programmation linéaire avec application aux graphes.* Presse polytechniques, 1990, Première edition.

[5] Maurras Jean F. : *Programmation linéaire, Complexité : Séparation et Optimisation.* Spinger-Verlag Berlin, Heidelberg, 2002

[6] Schrijver A. *Theory of linear and integer programming.* Wiley and Sons, 1986, (Cité page 6.)

[7] coursC2SI.pdf
`http://www.fsr.ac.ma/cours/maths/bernoussi/Cours%20C2SI.pdf`

[8] Revue d'Intelligence Artificielle, Vol : No.1999,
`http://www.info.univ-angers.fr/pub/hao/papers/RIA.pdf`

[9] Thèse : conception des métaheuristiques d'optimisation,
`https://tel.archives-ouvertes.fr/tel-01143778/document`

[10] THESE de ALAMI,
`http://homepages.laas.fr/elbaz/These_LALAMI.pdf`

[11] Interstices.info : Le problème de sac à dos,
`https://interstices.info/jcms/c_19213/le-probleme-du-sac-a-dos`

[12] Algorithme de cutting plane Coupe de Gomory - LITA
`http://www.lita.univ-lorraine.fr/~kratsch/teaching/ro10.ps`

[13] Integer programming : cutting planes, (à partir de la page 301).
`http://http://web.mit.edu/15.053/www/AMP-Chapter-09.pdf`

[14] REPERE2011,
Ressources Eléctroniques Pour les Etudiants, la Recherche et l'Enseignemen.
Paris, à jour le 01/06/2011, disponible sur :
`http://repere.enssib.fr`

[15] Fouad Bekkari, *[Mémoire de Magistère] : Résolution des problèmes difficiles par optimisation distribuée* UNIVERSITE MOHAMED KHIDER BISKRA, Algérie, 2008.