UNIVERSITE MOHAMED BOUDIAF M'SILA

FACULTE DES MATHEMATIQUES ET DE L'INFORMATIQUE

Département de Mathématiques

MEMOIRE DE FIN D'ETUDE

Présenté pour l'obtention du diplôme de **Master**

Domaine : Mathématiques et Informatique

Filière : Mathématiques

Option : Mathématiques discrètes

Par: Ben omara Hocine

Sujet

Méthodes exactes de solution de problème d'optimisation combinatoire: par séparation et évaluation et programmation dynamique

Dirigé par :

Mr. Mouhoub Nasseredine	Univerité de Bordj-Bouariridje	Président
Mr. Belouadah Hocine	Univerité de M'sila	Rapporteur
Mr. Bensaloua Cheniti	Univerité de Bordj-Bouariridje	Examinateur

Promotion: 2014/2015


بسم الله الرحمن الرحيم

Remerciements :

En touts premier lieu, je remercie Allah le tout puissant, et

Je tiens à remercier très sincèrement Pr. Belouadah Hocine pour le grand honneur qu'il m'a fait en me proposant le sujet, pour ses conseils et pour ses orientations J'ai eu l'honneur et le privilège de travailler sous son assistance et de profiter de ses qualités humaines m'ont été d'une grande utilité, Son professionnelles et de sa grande expérience. il m'a guidé tout au long de ce travail.

Je remercie tous les enseignants de l'université de M'sila, et en particulier mes enseignants de nos départements Mathématiques.

Mes remerciements vont également aux membres de jury d'avoir accepté de juger mon travail.

Je remercie vivement toute ma famille, en particulier mes parents, pour m'avoir toujours soutenu au cours de mes études. Qu'ils trouvent ici le fruit de leur patience et du soutien permanent qu'ils m'ont prodigué pour affronter tous les moments difficiles.

Mes sincères remerciements s'adressent également à mes amis Nasseredine, Aissa, Mohamade, Rabahe, Kamel Nadjoui,Khalile, et tous mes collègues.

Enfin, je remercie toutes les personnes qui de près ou de loin, ont contribue à sa réalisation de ce travail.

Ben amara Hocine

Table des Matières

Table des Matières

Table des Matières

Liste des Figures:

Liste des tableaux:

Introduction générale

Introduction générale

Dans notre mémoire on a considéré deux méthodes célèbres parmi les méthodes de résolution exacte de problèmes d'analyse combinatoire.

Ceux sont la méthode par séparation et évaluation et la méthode de programmation dynamique. Au premier chapitre nous avons introduit la notion de difficulté des problèmes combinatoires et la complexité des algorithmes associés. On a définit ceux qui sont polynomiaux et ceux qui ne sont pas. C'est d'ailleurs que les solutions des derniers qui nous intéressent.

Au deuxième chapitre on a considéré l'étude de la méthode par séparation et évaluation. En fait on a considéré celle qui est appliqué aux programmes linéaires à valeurs entières. Via un exemple on a essayé d'expliquer le mécanisme général de la méthode que trois facteurs majeurs déterminent son efficacité;

- La séparation: c'est la façon que la méthode divise l'espace de recherche pour situer mieux la solution optimale.
- L'évaluation par défaut (par excès) la plus serrée et proche de la valeur de la fonction optimale si son exécution nécessite peu de calcul est la meilleur.

- La stratégie de recherche: par largeur d'abord, par profondeur d'abord, le meilleur d'abord. Cette dernière semble la meilleure.

Le troisième chapitre est consacré à la programmation dynamique, cette méthode est basé sur le principe de Bellman: "toute sous-politique d'une politique optimale est optimale". Elle est toujours appliquée aux problèmes séquentiels. Dans notre cas on a abordé le problème d'ordonnancement de tâches sur une machine afin de minimiser le retard moyen.
D'ailleurs c'est selon la nature du problème en question que la méthode est utilisé récursivement que se soit en avant (forward) ou en arrière (backward). Une étude comparative avec l'énumération complète est inclue dans ce chapitre.
Il est important de noter que chacune de ces méthodes a ses inconvénients est ses avantages: contrairement à la programmation dynamique, la méthode par séparation et évaluation est moins rapide mais nécessite peu de mémorisation de données.

Chapitre I

Introduction

I.1 Introduction

L’optimisation combinatoire est un outil indispensable combinant diverses techniques des mathématiques discrètes et de l’informatique afin de résoudre des problèmes d’optimisation combinatoire de la vie réelle.

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.

Il s’agit, en général, de maximiser (problème de maximisation) ou de minimiser (problème de minimisation) une fonction objectif sous certaines contraintes. Le but est de trouver une solution optimale dans un temps d’exécution raisonnable.

L'optimisation combinatoire est l'une des plus jeunes branches et les plus actives des mathématiques discrètes de ces dernières années. Dans ce chapitre nous allons donner quelques définitions des notions de l'optimisation combinatoire et de stratégies d’extraction de problèmes et parmi elle la méthode par séparation et évaluation et la méthode de programmation dynamique sur les quelle nous allons nous pencher avec plus de détails.

Le mot " Combinatoire " désigne la discipline des mathématiques concernée par les structures " discrètes " ou " finies ". Citons quelques branches de cette discipline : la théorie des graphes, la combinatoire énumérative, les problèmes de dénombrement, la combinatoire polyèdrale, l'optimisation combinatoire, etc. Les frontières entre ces branches ne sont pas hermétiques, les différentes branches expriment plutôt des orientations méthodologiques différentes. L'" optimisation "ou programmation mathématique " sont des termes utilisés pour recouvrir toutes les méthodes qui servent à déterminer l'optimum d'une fonction avec (ou sans) contraintes. Cette fonction modélise le choix optimal. On optimise déjà à l'école, quand on apprend comment déterminer l'optimum (minimum ou maximum) d'une fonction différentiable. Les ingénieurs, les économistes, la nature, font souvent des choix optimaux (ou quasi-optimaux), d'où l'importance de l'optimisation dans les sciences, qu'elles soient techniques, mathématiques, physiques, informatiques, économiques, naturelles, etc [8].

L' **" Optimisation combinatoire "** consiste à trouver le meilleur entre un nombre fini de choix. Autrement dit, minimiser une fonction, avec contraintes, sur un ensemble fini. Quand le nombre de choix est exponentiel (par rapport à la taille du problème), mais que les choix et les contraintes sont bien structurés, des méthodes mathématiques doivent et peuvent intervenir, comme dans le cas " continu ", pour permettre de trouver la solution en temps polynomial (par rapport à la taille du problème). Alors, l’optimisation combinatoire consiste à développer des algorithmes polynomiaux et des

Théorèmes sur des structures discrètes qui permettent de résoudre ces problèmes. Une autre définition a été donne comme suit [8].
L'optimisation combinatoire est une problématique consistant, pour un ensemble de variables prenant des valeurs entières, à déterminer leurs instanciations, éventuellement sous contraintes, correspondant au maximum (ou minimum) d'une fonction de ces variables, OF, généralement appelée fonction objectif. L'ensemble des combinaisons pouvant être prises par ces variables constitue l'espace de recherche du problème. Une combinaison donnée est appelée une solution potentielle ou plus simplement une solution, alors que les instanciations des variables qui maximisent OF sont appelées les solutions optimales du problème. Dans le cas d'une optimisation combinatoire sous contrainte, on définit un ensemble de contraintes sur les variables rendant non valide un sous-ensemble de l'espace de recherche. L'optimisation combinatoire se situe au carrefour de la théorie des graphes, de la programmation mathématique, de l'informatique théorique (algorithmique et théorie de la complexité) et de la recherche opérationnelle [8].

Dans ce chapitre nous présentons certaines méthodes parmi les plus représentatives, développées pour résoudre les problèmes d'optimisation combinatoire à savoir la séparation et évaluation et la programmation dynamique [5].

I.2 Problème d'optimisation combinatoire

I.2.1 Formulation de problèmes d'optimisation combinatoire

Comme il existe différentes méthodes pour obtenir une solution optimale à un grand problème d'optimisation combinatoire dans un temps de calcul raisonnable, il est souvent préférable de reformuler le problème. Dans ce contexte il est souvent préférable d'augmenter le nombre de variables et de contraintes. Une bonne formulation du problème consisterait à trouver une fonction objective qui soit le plus approprié au problème original. Un problème d'optimisation combinatoire peut se formuler de la manière suivante: données + question.
1 - Données:
- un ensemble fini d'éléments ou configuration.

- une fonction de coût ou poids sur ces éléments.
2 - Question:
Déterminer n éléments de coût minimum ou de poids maximum, ... ?

Quelques exemples de problèmes d'optimisation combinatoire: Plus cours chemin, Arbre de poids minimum, Voyageur de commerce, Réseaux de transports, coloration d'un graphe, problème d'ordonnancement [8].

I.2.2 Définition:

Définition 1.1: Une fonction f (n) est O(g(n)) (f (n) est de complexité g(n)), s'il existe un réel c > 0 et un entier positif n_0 tel que pour tout $n \geq n_0$ on a $|f(n)| \leq c.g(n)$[3].

Définition 1.2: Un algorithme en temps polynomial est un algorithme dont le temps de la complexité est O(p(n)), où p est une fonction polynomiale et n est la taille de l'instance (ou sa longueur d'entrée).
Si k est le plus grand exposant de ce polynôme en n, le problème correspondant est dit être résoluble en $O(n^k)$ et appartient à la classe P, un exemple de problème polynomial est celui de la connexité dans un graphe [3].

Définition 1.3: La classe NP contient les problèmes de décision qui peuvent être décidés sur une machine non déterministe en temps polynomial. C'est la classe des problèmes qui admettent un algorithme 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 énumérant l'ensemble de solutions possibles et en les testant à l'aide d'un algorithme polynomial [3].

Définition1.4: On dit qu'un problème de recherche P_1 se réduit polynomialement à un problème de recherche P_2 par la réduction de Turing s'il existe un algorithme A_1 pour résoudre P_1 utilisant comme sous programme un algorithme A_2 résolvant P_2, de telle sorte que la complexité de A_1 est polynomiale, quand on évalue chaque appel de A_2 par une constante [3].

Définition 1.5: La classe NP-complet: parmi l'ensemble des problèmes appartenant à NP, il en existe un sous-ensemble qui contient les problèmes les plus difficiles: on les appelle les problèmes NP-complets. Un problème NP-complet possède la propriété que tout problème dans NP peut être transformé (réduit) en celui-ci en temps polynomial. C'est à dire qu'un problème est NP-complet quand tous les problèmes appartenant à NP lui sont réductibles. Si on trouve un algorithme polynomial pour un problème NP-complet, on trouve

alors automatiquement une résolution polynomiale de tous les problèmes de la classe NP [3].

Définition 1.6: 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 existe un problème NP-complet se réduisant à ce problème par une réduction de Turing [3].

I.4 Problème de reconnaissance

Un problème de reconnaissance est un problème qui consiste à apporter une réponse "oui" ou "non" à une question.
A chaque problème d'optimisation combinatoire, on peut associer un problème de reconnaissance de la manière suivante.

Soit un problème d'optimisation combinatoire:
Trouver $s' \in S \mid f(s') = \min \{f(s) \mid s \in S\}$.
Soit a un nombre, on définit le problème de reconnaissance associé:
Existe-t-il $s' \in S \mid f(s') \leq a$?

Un problème d'optimisation combinatoire est au moins aussi difficile que le problème de reconnaissance associé. De plus, on peut généralement prouver que le problème de reconnaissance n'est pas plus facile que le problème d'optimisation combinatoire. En d'autres termes, cela signifie qu'un problème d'optimisation combinatoire est souvent du même niveau de difficulté que le problème de reconnaissance associé. Cela justifie que la suite de ce chapitre ne concerne que les problèmes de reconnaissance [10].

I.5 Problème facile, difficile

I.5.1 Problèmes décidables

Tout d'abord, on fait une distinction entre les problèmes décidables et les problèmes indécidables. Les problèmes indécidables sont ceux pour lesquels aucun algorithme, quel qu'il soit, n'a été trouvé pour les résoudre. Ainsi, les problèmes décidables sont ceux pour lesquels il existe au moins un algorithme pour les résoudre [10].

I.5.2 La classe NP

Parmi les problèmes décidables, les plus simples à résoudre sont regroupés dans la classe NP. Un problème appartient à la classe NP si quelqu'un ayant la solution au problème peut démontrer que c'est la solution en un temps polynomial. Les autres problèmes décidables sont considérés comme très difficiles. La classe NP est également décomposée en trois catégories qui

permettent d'identifier les problèmes les plus simples et les problèmes les plus compliqués de la classe [10].

I.5.3 La classe P

La classe P, qui regroupe les problèmes les plus simples de la classe NP, contient les problèmes pour lesquels on connaît au moins un algorithme polynomial pour les résoudre. Pour le reste de la classe NP,
on n'est pas sûr qu'il n'existe pas un algorithme polynomial pour résoudre chacun de ses problèmes. Ainsi, on sait que P est inclu dans NP mais on n'a pas pu prouver que P n'est pas NP [10].

I.5.4 Réduction polynomial

Soit deux problèmes P_1 et P_2. On dit que P_1 est réduit au problème P_2 si on peut résoudre P_1 en utilisant un algorithme pour P_2 comme sous-routine. Cette réduction est dite polynomiale si l'algorithme pour P_1 est polynomial en comptant l'appel à la sous-routine de P_2 comme une opération élémentaire.
P_2 est au moins aussi difficile que P_1. En effet, si P_2 appartient à la classe P, P_1 y appartient aussi, car l'algorithme ci-dessus est polynomial. Si P_2 n'est pas polynomial, rien n'empêche P_1 de l'être s'il existe un algorithme polynomial pour le résoudre [10].

I.5.5 La classe NP-Complet

La classe NP-Complet regroupe les problèmes les plus difficiles de la classe NP. Elle contient les problèmes de la classe NP tels que n'importe quel problème de la classe NP leur est polynomialement réductible. Entre eux, les problèmes de la classe NP-Complet sont aussi difficiles [10].

I.5.6 La classe NP-Difficile

La classe NP-Difficile regroupe les problèmes (pas forcément dans la classe NP) tels que n'importe quel problème de la classe NP leur est polynomialement réductible [10].

I.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. Ceci ne peut être fait qu'avec une bonne connaissance du problème sous étude et de son domaine d'application.
Les méthodes d'optimisation peuvent être réparties en deux grandes classes de méthodes pour la résolution des problèmes :
Les méthodes approchées et les méthodes exactes, C'est d'ailleurs que sur ces dernières qu'on va se pencher [5].

I.3 Les méthodes exactes

Les algorithmes exacts sont utilisés pour trouver au moins une solution optimale d'un problème. Les algorithmes exactes les plus réussis dans la l'itération sont:
La méthode par séparation et évaluation et la méthode de programmation dynamique [5].

I.3.1 La méthode par séparation et évaluation

La méthode par séparation et évaluation (procédure par séparation et évaluation 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 séparation et évaluation dépend, entre autres, de la qualité de cette fonction (de sa capacité d'exclure des solutions partielles tôt) [5].

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

Figure I.1 – Algorithme général de séparation et évaluation

Comme application de cette méthode au problème linéaire à valeurs entières l'algorithme devient:

Entrée: Problème de minimisation R
Sortie: x^* la solution optimale et z^* la valeur optimale (si $z^* = 1$, alors R n'a pas de solution réalisable) [7].

Initialisation
1: L ← {R}
2: $z^* = \infty$

Critère d'arrêt
3: Si L = ϕ alors x^* est la solution optimale et z^* la valeur optimale.

Arrêter Sélection du nœud
4: Choisir Q ∈ L
5: L L \ {Q}

Évaluation
6: Résoudre la relaxation $\tilde{Q}$ de Q
7: Si $\tilde{Q}$ est non réalisable alors $\tilde{z} \leftarrow \infty$ sinon poser $\tilde{x}$ la solution optimale de $\tilde{Q}$ et $\tilde{z}$ sa valeur optimale

Élagage
8: Si $\tilde{z} \leftarrow z^*$ alors aller à la ligne 3

Vérification

9: Si $\tilde{x}$est une solution réalisable pour R alors $x^* \leftarrow \tilde{x}$, $z^* \leftarrow \tilde{z}$ et aller à la ligne 3

Branchement

10: Séparer Q en des sous-problèmes tel que $F(Q) = F(Q_1) \cup \ldots\ldots \cup F(Q_k)$

11: $L \leftarrow L \cup \{Q1, \ldots, Q_k\}$

12: Aller à la ligne 3

Figure I.2 – Algorithme par séparation et évaluation

I.3.2 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 [5].

I.3.2.1 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 [5].

I.6 **Conclusion:**

Dans ce chapitre on parlé des problèmes d'optimisation combinatoire de leurs degrés de difficulté. Aussi des algorithme associes et de leurs complexités. Puis on s'est penché sur les deux méthodes de solution exactes. Dans les chapitres qui suivent on va considérer deux fameux problèmes d'optimisation combinatoire: le premier est le problème linéaire à valeur entières et le deuxième est le problème d'ordonnancement d'atelier qui est l'ordonnancement de tâches sur une machine afin de minimiser le retard moyen. Pour le premier on va appliquer la méthode par séparation et évaluation pour déterminer une solution exacte. On va expliquer le mécanisme et le fonctionnement de la méthode. Quant au deuxième on va appliquer la méthode de programmation dynamique avec une étude théorique et numérique on va confirmer sa superiorité sur l'énumération complète.

Chapitre II

La méthode par
séparation et évaluation

II.1 **Introduction**

L'algorithme par séparation et évaluation est une méthode qui permet d'énumérer implicitement l'ensemble des solutions réalisables d'un problème de programmation linéaire en nombres entiers. Il suit le principe de « diviser pour régner » en décomposant récursivement un problème en plusieurs sous-problèmes plus simples à résoudre. La solution optimale du problème est retrouvée en prenant la solution du meilleur sous-problème, c'est-à-dire le sous-problème qui permet d'identifier la solution réalisable ayant la plus petite borne.

II.2 **La méthode par séparation et évaluation (Branch and Bound)**

L'algorithme par séparation et évaluation, plus connu sous son appellation anglaise Branch and Bound (B&B) (Land et Doig 1960), repose sur une méthode arborescente de recherche d'une solution optimale par séparations et évaluations, en représentant les états solutions par un arbre d'états, avec des nœuds, et des feuilles.
La séparation et évaluation est basé sur trois axes principaux :
– La séparation,
– L'évaluation,
– La stratégie de recherche.

II.2.1 **La séparation**

La séparation consiste à diviser le problème en sous-problèmes.
Ainsi, en résolvant tous les sous-problèmes et en gardant la meilleure solution trouvée, on est assuré d'avoir résolu le problème initial. Cela revient à construire un arbre permettant d'énumérer toutes les solutions. L'ensemble de nœuds de l'arbre qu'il reste encore à parcourir comme étant susceptibles de contenir une solution optimale, c'est-à-dire encore à diviser, est appelé ensemble des nœuds actifs.

II.2.2 **L'évaluation**

L'évaluation permet de réduire l'espace de recherche 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. La séparation et évaluation 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 de plus bas coût rencontré pendant l'exploration, et à comparer le coût de chaque nœud parcouru à celui de la meilleure

solution. Si le coût du nœud considéré est supérieur au meilleur coût, on arrête l'exploration de la branche et toutes les solutions de cette branche seront nécessairement de coût plus élevé que la meilleure solution déjà trouvée.

II.2.3 **La stratégie de recherche**

II.2.3.1 **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 autres stratégies présentées.

II.2.3.2 **La profondeur d'abord:**

Cette stratégie avantage les sommets les plus éloignés de la racine (de profondeur la plus élevée) en appliquant plus de séparations au problème initial. Cette voie mène rapidement à une solution optimale en économisant la mémoire.

II.2.3.3**Le meilleur d'abord:**

Cette stratégie consiste à explorer des 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 [3].

II.3 Séparation et évaluation pour les problèmes linéaires à variable entiers:

– Les méthodes par séparation et évaluation sont des méthodes basées sur une énumération "intelligente" des solutions admissibles d'un problème d'optimisation combinatoire.

– Idée : prouver l'optimalité d'une solution en partitionnant l'espace des solutions.

– "Diviser pour régner"

– Application à la programmation linéaire en nombres entiers : utilise toute la puissance de la programmation linéaire pour déterminer de bonnes bornes.

– On appelle relaxation linéaire d'un programme linéaire en nombres entiers le programme linéaire obtenu en supprimant les contraintes d'intégralité sur les variables [4].

II.3.1 **Programme en nombres entiers**

$$\begin{array}{lll}(P) & \max & c^T x \\ & \text{s.c.} & Ax \leq b \\ & & x \geq 0, \text{ entier.}\end{array}$$

II.3.2 **Relaxation linéaire**

$$\begin{array}{lll}(LP) & \max & c^T x \\ & \text{s.c.} & Ax \leq b \\ & & x \geq 0.\end{array}$$

II.3.3 **Propriétés de la relaxation linéaire**

– La valeur de la solution optimale de LP est une borne supérieure sur la valeur de la solution optimale de P.
– La valeur d'une solution admissible de P fournit une borne inférieure sur la valeur de la solution optimale de P.
– Si la solution optimale de LP est entière (donc admissible pour P), elle est également la solution optimale de P [4].

II.3.4 **Branchement**

– Si la solution de LP n'est pas entière, soit x_i une variable prenant une valeur fractionnaire x_i^* dans la solution optimale de LP.
– Le problème peut être divisé en deux sous-problèmes en imposant

$$x_i \leq \lfloor x_i^* \rfloor \qquad \text{ou} \qquad x_i \geq \lfloor x_i^* \rfloor + 1$$

Où $\lfloor x_i^* \rfloor$ est le plus grand entier inférieur à x_i^*

– La solution optimale de P est la meilleure des solutions optimales des deux problèmes

$$\begin{array}{lll}(P_1) & \max & c^T x \\ & \text{s.c.} & Ax \leq b \\ & & x_i \leq \lfloor x_i^* \rfloor \\ & & x \geq 0, \text{ entier.}\end{array} \qquad \begin{array}{lll}(P_2) & \max & c^T x \\ & \text{s.c.} & Ax \leq b \\ & & x_i \geq \lfloor x_i^* \rfloor + 1 \\ & & x \geq 0, \text{ entier.}\end{array}$$

II.4 **Exemple (séparation et évaluation).**

(P)

$$\begin{array}{rrrrl} \max z = & 5x_1 & +4x_2 & & \\ \text{s.c.} & x_1 & +x_2 & \leq 5 & \\ & 10x_1 & +6x_2 & \leq 45 & \\ & x_1, & x_2 & \geq 0, \text{ entiers.} & \end{array}$$

Solution de

$$(LP) \quad x_1 = 3.75,\ x_2 = 1.25$$
$$z = 23.75$$

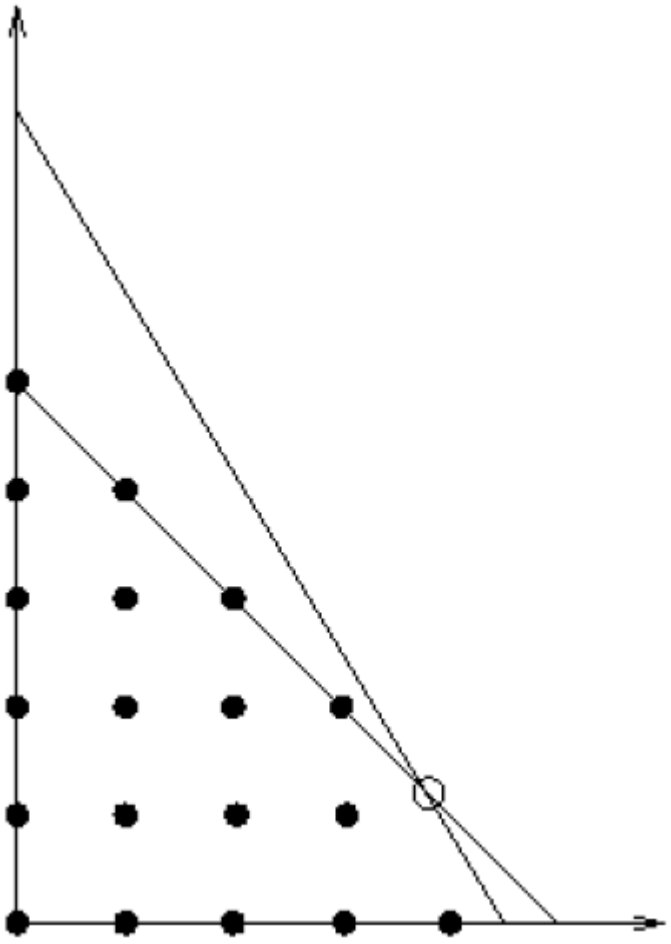
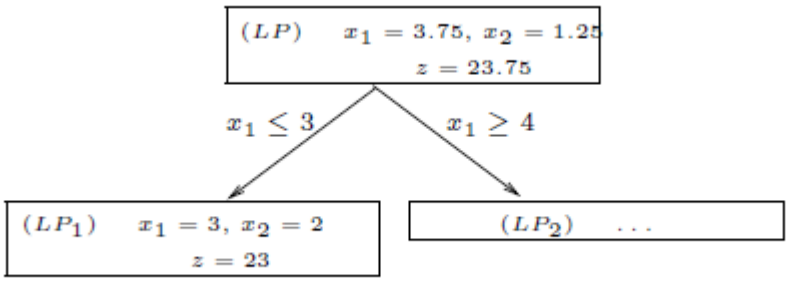


Figure **II**.1 l'ensemble de solution admissible pour le problème (P)



– La solution de LP_1 est une solution admissible de P et donc z = 23 est une borne inférieure sur la valeur de la solution optimale de P.

– Le neoud correspondant peut être éliminé vu qu'une solution entière optimale satisfaisant $x_1 \leq 3$ a été trouvée (solution de P_i)

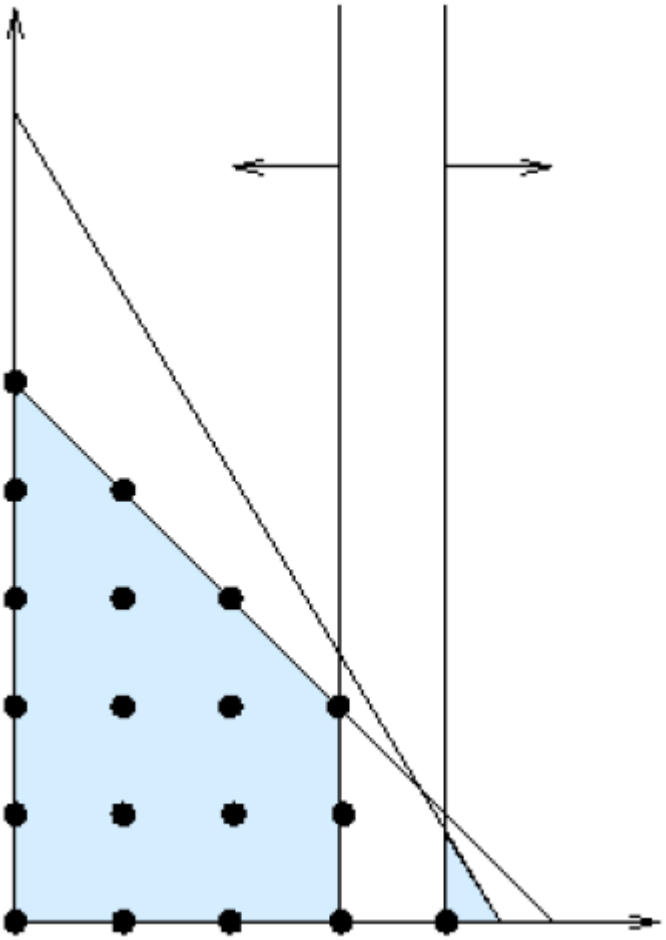


Figure **II**.2 séparation de l'ensemble de solution admissible pour le problème (P)

– La valeur de la solution de LP, z = 23.75 est une borne supérieure sur la valeur de la solution optimale de P.

– Vu que tout les coefficients sont entiers, on peut en déduire que la valeur de la solution optimale de P est inférieure ou égale à 23.
– La solution de P_1 est donc optimale pour P.

II.5 **Règles de branchement**

– Il n'y a pas de règle générale pour le choix de la variable de branchement et de la branche à examiner en premier.
– Ce choix peut avoir un impact important sur le nombre de neouds à examiner dans l'arbre de séparation et évaluation.
– Exemple: branchement d'abord du côté $\geq$ [4].

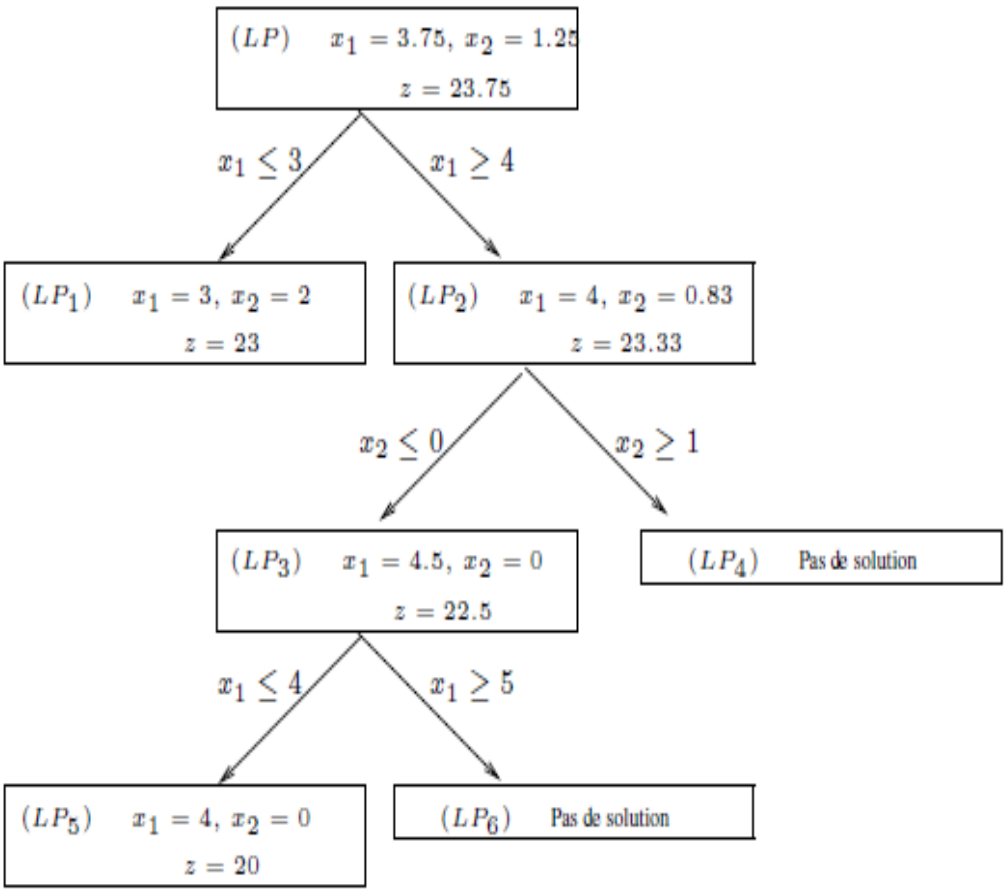


Figure **II**.3 L'arbre de séparation et évaluation.

II.6 **Conclusion:**

Il est important de noter que la méthode par séparation et évaluation n'est pas une méthode spécifique appliqué à un problème spécifique mais plutôt une méthode générale qui pourrait être appliquer à des problèmes différents. C'est plutôt l'évaluation par excès (défaut) la séparation et la stratégie de recherche qui changent selon le problème abordé. Prenons par exemple le problème d'ordonnancement de tâches avec dates de disponibilités sur une machine afin de minimiser la somme pondérée des dates de fin $1/r_i//\sum w_i C_i$. Pour appliquer la méthode par séparation et évaluation on utilise comme évaluation par défaut la valeur de la solution optimale du problème $1//\sum w_i C_i$ (relâché: $r_i = 0$) qui est déterminée par la règle SWPT (shortest weighted processing time). Quant au séparation: au niveau 1 de l'arbre de la méthode par séparation est évaluation il y' aura n nœuds fils chacun correspond à la 1^{er} position de de la tâche. Et par conséquent tout nœud i fils correspond à l'ensemble de séquence ayant la $i^{ième}$ tâche à la 1^{er} position. Quant à la stratégie de recherche on peut utiliser n'importe qu'elle stratégie des trois connues.

Chapitre III

La méthode de programmation dynamique

III.1 **Introduction**

La programmation dynamique est une approche de solution, de problème souvent complexe et difficile à résoudre qui permit à le décomposer à un nombre de problèmes plus petits qui sont souvent moins difficiles à résoudre. De plus, la programmation dynamique nous permit de diviser un grand problème de telle manière que lorsque tous les sous-problèmes sont résolus, on finit par une solution optimale du problème global. On va voir que chacun de ces sous-problèmes créer est identifié par une stade de la procédure de programmation dynamique. Par conséquent cette technique a été appliquée à plusieurs problèmes de décisions qui sont de multiple stade en nature. Citons par exemple le problème d'achat le problème de plus court chemin.

III.2 **L'approche de Held et Karp:**

En 1962 Held et Karp ont inspiré du principe de Bellman l'idée d'appliquer la programmation dynamique au problème d'ordonnancement sue une machine unique où l'objectif prend la forme $\sum_i \gamma(C_i)$ où $\gamma(C_i)$ est une mesure régulière. Ici on essaye d'appliquer cette approche au problème $1//\overline{T}$

(ici $\gamma(C_i) = \frac{1}{n} \max (C_i - d_i , 0)$) cette approche est basé sur la simple remarque suivante: concernant la solution d'un ordonnancement optimal: ceci dit que dans un ordonnancement optimal les premières k tâches

(k = 1,….,n) doivent former un ordonnancement optimal pour le problème réduit basé sur ses k tâches.

Dorénavant on note par $\{J_1,\ldots,J_n\}$ l'ensemble des n tâches sans ordre donné. Alors qu'on note $(J_1,\ldots,J_n)$ la séquence de n tâches où J_1 est ordonnancé 1^{re}, J_2 est ordonnancé, $2^{ième}$,…,J_n est ordonnancé $n^{ième}$ **[6]**.

III.2.1 **Programmation dynamique en avant (forward):**

Si Q est un ensemble quelconque de tâches contenant la tâche J_i , alors

Q - $\{J_i\}$ est l'ensemble des tâches obtenus par suppression de J_i de Q.
Aussi pour l'ensemble Q on définit C_Q par:

$$C_Q = \sum_{J_i \text{ in } Q} p_i$$

Donc C_Q est la somme des temps d'exécution de toutes les tâches dans Q. Puisqu' on considère un critère $\overline{T}$ qui est croissante avec les C_i, alors C_Q n'est rien d'autre que la date de fin de la dérnière tâche ordonnancée. Après on défini Γ(Q) comme étant le coût minimal obtenu par ordonnancer les tâches dans Q optimallement.

Si Q contient une seule tâche, soit Q = $\{J_i\}$, alors

$$\Gamma(Q) = \Gamma(\{J_i\}) = \gamma_i(p_i) \qquad (1)$$

Puisqu' il y' a une seule manière d'ordonnancer une tâche alors celle-ci se termine à $C_i = p_i$. Supposons maintenant que Q contient K > 1 tâches. Alors, sachant que la dernière tâche doit compléter à C_Q et que dans n'importe quelle séquence optimale les (K – 1) premières tâches sont ordonnancées optimallement pour un problème réduit contenant seulement ces tâches-ci, on a:

$$\Gamma(Q) = \min_{J_i \in Q}\{\Gamma(Q - \{J_i\}) + \gamma_i(C_Q)\} \qquad (2)$$

Autrement dit, pour trouver le coût minimal d'ordonnancement des taches de Q on considère chaque tâche à part et demanter combien elle coûte de l'ordonnancer à la fin. On trouve notre réponse en prenant le minimum de toutes ces possibilités.

Maintenant (1) définit Γ (Q) pour tout Q contenant une seule tâche. En utilisant ces valeurs et la relation (2) on peut trouver Γ(Q) pour tout Q contenant uniquement deux tâches; puis pour tout Q contenant uniquement trois tâches; etc…jusqu'à l'aboutissement à $\Gamma(\{J_1, J_2, \ldots, J_n\})$.

En faisant ça on détermine aussi une solution optimale. Peut être il est plus facile de voir tout ça dans un contexte d'un exemple particulier [9].

III.2.2 **Application:**

Exemple: Résoudre le problème 4/1//$\overline{T}$ avec les données:

Tâche J_i	J_1	J_2	J_3	J_4
Temps d'exécution p_i	8	6	10	7
Date échue d_i	14	9	16	16

Ici $\gamma_i(C_i) = \frac{1}{4} \max\{C_i - d_i, 0\}$ pour tout i = 1,2,3,4.

Tout d'abord, on calcule $\Gamma(Q)$ pour les quatre ensembles qui contiennent chacune une seule tâche.

Par exemple en utilisant (1) on a:

$$\Gamma(\{J_1\}) = \gamma_1(C_1) = \frac{1}{4}\max\{0, C_1 - d_1\} = \frac{1}{4}\max\{0, p_1 - d_1\}$$

Donc on génère:

Table III.1- Γ (Q) pour les quatre ensembles réduit à une seule tâche

Q	$\{J_1\}$	$\{J_2\}$	$\{J_3\}$	$\{J_4\}$
$p_i - d_i$	-6	-3	-6	-9
$\Gamma(Q)$	0	0	0	0

Puis on calcule $\Gamma(Q)$ pour les six ensembles contenants uniquement deux tâches chacune.
Par exemple**,** si $Q = \{J_1, J_2\}$, on a $C_Q = p_1 + p_2 = 14$ et par (2)

$\Gamma(Q) = \min\{ \Gamma(\{J_1\}) + \gamma_2(14), \Gamma(\{J_2\}) + \gamma_1(14)\}$

$= \min\{ 0 + 5/4, \quad 0 + 0 \}$

$= 0.$

Les calculs pour les sous-ensembles de deux tâches sont donnés dans la table suivante.

Table III.2 Γ (Q) pour chaque sous-ensemble de deux tâches

Q	$\{J_1, J_2\}$	$\{J_1, J_3\}$	$\{J_1, J_4\}$	$\{J_2, J_3\}$	$\{J_2, J_4\}$	$\{J_3, J_4\}$
C_Q	14	18	15	16	13	17
J_i tâche dernière dans la séquence	J_1 J_2	J_1 J_3	J_1 J_4	J_2 J_3	J_2 J_4	J_3 J_4
$\gamma_i(C_Q)$	0 $\frac{5}{4}$	1 $\frac{1}{2}$	$\frac{1}{4}$ 0	$\frac{7}{4}$ 0	1 0	$\frac{1}{4}$ $\frac{1}{4}$
$\Gamma(Q-\{J_i\})+\gamma_i(C_Q)$	0 $\frac{5}{4}$	1 $\frac{1}{2}$	$\frac{1}{4}$ 0	$\frac{7}{4}$ 0	1 0	$\frac{1}{4}$ $\frac{1}{4}$
Minimum	*	*	*	*	*	*
Γ (Q)	0	$\frac{1}{2}$	0	0	0	$\frac{1}{4}$

Il y à deux possibilités pour chaque Q correspondant à quelle tâche est ordonnancé dernière: les colonnes sont divisés en conséquence. On marque la colonne correspondante au minimum par un astérisque. Maintenant on calcule Γ (Q) pour quatre ensembles contenant chacune trois tâches.

Par exemple:

Si $Q = \{ J_1, J_2, J_3 \}$, on a $C_Q = p_1 + p_2 + p_3 = 24$ et par (2)

$\Gamma(Q) = \min \{\Gamma(\{J_1, J_2\}) + \gamma_3(24), \Gamma(\{J_1, J_3\}) + \gamma_2(24), \Gamma(\{J_2, J_3\}) + \gamma_1(24)\}$

$= \min \{0 + 2{,}1/2 + 15/4, 0 + 10/4\}$
$= 2.$

Les calculs pour les autres sous-ensembles de trois tâches sont donnés dans la table **III**.3.

Table III.3 Γ (Q) pour chaque sous-ensemble de trois-tâches

Q	$\{J_1, J_2, J_3\}$	$\{J_1, J_2, J_4\}$	$\{J_1, J_3, J_4\}$	$\{J_2, J_3, J_4\}$
C_Q	24	21	25	23
J_i tâche finale dans la séquence	J_1 J_2 J_3	J_1 J_2 J_4	J_1 J_3 J_4	J_2 J_3 J_4
$\gamma_i(C_Q)$	$\frac{10}{4}$ $\frac{15}{4}$ 2	$\frac{7}{4}$ 3 $\frac{5}{4}$	$\frac{11}{4}$ $\frac{9}{4}$ $\frac{9}{4}$	$\frac{14}{4}$ $\frac{7}{4}$ $\frac{7}{4}$
$\Gamma(Q-\{J_i\})+\gamma_i(C_Q)$	$\frac{10}{4}$ $\frac{17}{4}$ 2	$\frac{7}{4}$ 3 $\frac{5}{4}$	3 $\frac{9}{4}$ $\frac{11}{4}$	$\frac{15}{4}$ $\frac{7}{4}$ $\frac{7}{4}$
Minimum	*	*	*	*
Γ (Q)	2	$\frac{5}{4}$	$\frac{9}{4}$	$\frac{7}{4}$

Finalement on calcule Γ (Q) pour l'ensemble contenant tous les quatre tâches $Q = \{J_1, J_2, J_3, J_4\}$. Ici on a $C_Q = p_1 + p_2 + p_3 + p_4 = 31$. On calcule

Γ (Q) dans la table **III**.4. (Les calculs sont similaires à ceux des autres tables).

Table III.4 Γ (Q) pour l'ensemble entier des quatre tâches

Q	$\{J_1 , J_2 , J_3 , J_4\}$
C_Q	31
J_i tâche finale dans la séquence	J_1 J_2 J_3 J_4
$\gamma_i (C_Q)$	$\frac{17}{4}$ $\frac{22}{4}$ $\frac{15}{4}$ $\frac{15}{4}$
$\Gamma (Q-\{J_i\}) + \gamma_i (C_Q)$	6 $\frac{31}{4}$ 5 $\frac{23}{4}$
Minimum	*
Γ (Q)	5

Ainsi on a montré que l'ordonnancement de quatre tâches optmallement donne un minimum de retard moyenne égale à 5. En plus il est facile de déterminer un ordonnancement optimal. L'astérisque dans la première colonne dans la table, montre que le retard moyen minimal est obtenu quand J_3 est ordonnancer dernière. Ceci veut dire que $\{J_1 , J_2 , J_4\}$ sont les trois premières tâches qui doivent être exécutés.
Donc en regardant à la table (III.3) on trouve que J_4 doit être ordonnancer dernière parmi ces trois tâches par conséquent on sait que $\{J_1 , J_2\}$ sont les deux premières tâches qui doivent être exécutés. En regardant à la table (III.2), on trouve que J_1 doit être exécuté dernière. Ceci laisse J_2 comme la tâche ordonnancée première [9].

Donc une solution optimale est (J_2 , J_1 , J_4 , J_3).

III.2.3 **Calcul de retard:**

III.2.3.1 **Algorithme 2.1**

Etape 1. Poser $k = 0$, $C_{i(0)} = 0$, $\Sigma = 0$.

Etape 2. Augmenter k par

Etape 3. $C_{i(k)} = C_{i(k-1)} + p_{i(k)}$; $L_{i(k)} = C_{i(k)} - d_{i(k)}$; $T_{i(k)} = \max\{ 0, L_{i(k)}\}$

Etape 4. Ajouter $T_{i(k)}$ to Σ

Etape 5. Est ce que K = n? Si c'est oui arrêter. Sinon aller à Etape 2.

For the schedule ($J_{i(1)}, J_{i(2)}, \ldots, J_{i(n)}$)

n additions pour augmenter k via k = 1, 2, …, n

n additions pour calculer les dates de fin

n soustractions pour calculer le retard relatif

n comparaisons pour calculer le retard

n additions pour cumuler la somme Σ

n comparaisons pour voir si k = n

___

6n

Donc pour calculer le retard total pour chacune des n! ordonnancements possibles il faut 6n opérations.

Maintenant pour déterminer le retard total minimal on utilise un algorithme qui se ressemble au suivant:

III.2.4 **Choix de meilleur retard moyen par énumération complète**

III.2.4.1 **Algorithme 2.2**

Etape 1. Poser $\sum_{min} = \Sigma$, μ = 1.

Etape 2. Augmenter j par 1.

Etape 3. Est-ce que $\sum_{min} \leq \sum_{i}$? Si c'est oui allen à Etape 5.

Etape 4. Affecter $\sum_{min}$ à Σ_i et affecter μ à j.

Etape 5. Est-ce que j = (n!) ? Si c'est oui arrêter. Sinon aller à Etape 2.

Le nombre d'opérations de cet algorithme est égal à:

(n! – 1) additions pour augmenter j via j =1, 2,…, n!

(n! – 1) comparaisons pour déterminer le retard total minimal

(n! – 1) comparaisons pour voir si j = n!

_____

3(n! – 1)

Donc l'énumération complète nécessite

$$6n\,(n!) + 3(n! - 1) \qquad (3)$$

opérations pour résoudre le problème $n/1//\overline{T}$.

Considérons maintenant la méthode de programmation dynamique:

Cette méthode calcule $\Gamma(Q)$ pour

$\binom{n}{1}$ Sous-ensemble Q de taille 1,

$\binom{n}{2}$ Sous-ensemble Q de taille 2,

$\binom{n}{3}$ Sous-ensemble Q de taille 3,

……

……

et $\binom{n}{n}$ Sous-ensemble Q de taille n.

L'algorithme 2.1ci-dessus à basin d' execution les operation mathematique suivantes pour chaque ensemble Q.

Addition pour calculer C_Q

K Soustractions pour calculer le retard relatif pour chaque dernière tâche possible

$$L_i = C_Q - d_i .$$

K Comparisons pour calculer le retard pour chaque dernière tache possible

$$T_i = \max \{L_i , 0\}.$$

K Additions pour former chaque somme : $\Gamma(Q - \{J_i\}) + \gamma_i (C_Q)$.

K Comparaisons pour déterminer le minimum de ces sommes, $\Gamma(Q)$

2K Opérations pour controller une boucle

———

6K + 1

Le passage d'un sous-ensemble à un outré necessities deux opérations pour chaque sous-ensemble Q. Par consequent la programmation dynamique necessities (6K+3) operations pour chaque sous-ensemble Q de taille K. Au total, done, elle necessilie:

$$\binom{n}{1}(6.1 + 3) + \binom{n}{2}(6.2 + 3) + \binom{n}{3}(6.3 + 3) + \ldots + \binom{n}{n}(6n + 3)$$

operations. C'est à dire:

$$6n.\ 2^{n-1} + 3(2^n - 1). \qquad (4)$$

III.2.5 **Comparaison entre l'énumération complète et la programmation dynamique:**

Maintenant nous sauves en mesure de comparer le nombre d'opérations nécessaires pour l'numération complète avec celles que nécessite de la programmation dynamique.

Le tableau ci-dessous montre les valeurs de expressions (3) et (4) pour des différents valeurs de n. supposons qu'on utilise un ordinateur de vitesse 1microscconde pour exécuter une opération. Donc l'numération complète dure 10 milles d'années pour résoudre un 20/1//T problème (d'quai la 3ligne de tableau), alors que la programmation dynamique resoud ce problème en 1 minute. Ceci montre la rapidité de la programmation dynamique. Malheureusement, cette méthode nécessite beaucoup de mémorisation. Elle doit mémoriser pulsions opération inter médianes. Par exemple elle doit se sourener chaque tâche ordonnancée dernière dans chaque Q et sa valeur $\Gamma(Q)$. Il n'a 2^n valeur. Malheureusement les ordinateurs ont de mémoires limités pas plus de 2^{30} [6].

Table III.5 Comparaison des calculs nécessaires pour la résolution de n/1// $\overline{T}$ par énumération complète et par la programmation dynamique

Nombre d'opérations nécessaires par		
n	Enumération complète	Programmation dynamique
4	647	237
10	2.286×10^{8}	33789
20	2.992×10^{20}	6.396×10^{7}
40	1.983×10^{50}	1.352×10^{14}

III.2.6 **Programmation dynamique en arriéré (backward):**

Pour un ensemble de tâche Q on définit: $s_Q = \sum_{J_i \notin Q} P_{J_i}$

ceci n'est rien d'autre que la date de plutôt pour n'importe quelle tâche de Q commence si toutes les tâches hors Q doivent être exécuter les premières.

On définit aussi Δ(Q) le coût minimum résultant de l'ordonnancement des tâches dans Q sones la condition qu'aucune ne peut commencer avant s_Q.

Pour $Q = \{J_i\}$ on a:

$\Delta(Q) = \Delta\{J_i\} = \gamma(s_{\{Ji\}} + p_i)$.

Car la tâche unique J_i doit finir à $(s_{\{Ji\}} + p_i)$.

$$\Delta(\{J_i\}) = \gamma(\sum_{j=1}^{n} p_j) \qquad (5)$$

Généralement, on trouve Δ(Q) en considérant chaque tâche dans Q et en calcula nr le coût d'ordonnancement de cette tâche.

Δ(Q) est les minimums des coûts:

$$\Delta(Q) = \min_{J_i \in Q}\{ \Delta(Q - \{J_i\}) + \gamma_i(s_Q + p_i)\} \qquad (6)$$

En utilisant (5) et (6) on construit les valeurs de $\Delta(Q)$ pour tous les ensembles Q.

III.3 **Conclusion:**

Il est important de noter que la méthode de Held et Karp pourrait être appliquer au problème d'ordonnancement sur une machine ou l'objectif est de la forme $\max_i \{\gamma(C_i)\}$.

Bibliographie

Bibliographie

[1] Mohamed Ali ALOULOU, "Mémoire Introduction aux problèmes d'ordonnancement", Universitè Paris Dauphine, 28 November 2005.

[2] Michel Bierlaire," Recherche Opérationnelle",2012.

[3] Sidi Mohamed. Douiri, Souad. Elbernoussi, Halima. Lakhbabk, "Cours des méthodes de résolution exactes heuristiques et métaheuristique" Université Mohammed V.

[4] Bernard. Fortz," Recherche opérationnelle et applications", 2012-2013, Page 39-41.

[5] Hacene. halim,"Mémoire Problème du l'optimisation combinatoire", Université Mohammed V,2011.

[6] Held.M. & Karp,R,M,"A dynamic programming approach to Sequencing problem ",journal SIAM, Value 10,pp196-210.

[7] Mathieu Larose,"Mémoire Développement d'un algorithme de branch-and-price-and-cut pour le problème de conception de réseau avec coûts fixes et capacities", Université de Montréal, 1er juin 2012.

[8] Rédha. Loucif, " Mémoire Parallélisation d'algorithmes d'optimisation Combinatoire"; Magister, Université conloinel hadj lakhdar- batna 13/01/2014.

[9] Rellman.R, "dynamic programming", Princeton University Press.

[10] Copyright (c) 1999-2000 - Bruno Bachelet - bachelet@ifrance.com - http://bruno.bachelet.net/

تم بحمد الله

Résumé:

Une étude numérique pour les deux méthodes de résolution exactes des problèmes d'analyse combinatoire est faite la méthode par séparation évaluation et la méthode de programmation dynamique. Pour la première on a pris comme méthode celle qui est appliqué, à la résolution des programmes linéaires à valeurs entières alors que le deuxième est celle appliqué aux problèmes séquentiels à savoir celle de d'ordonnancement des tâches sur une machine afin de minimiser leur retard moyen.

Mots-clés: séparation et évaluation, programmation dynamique, ordonnancement, programmation linéaire, analyse combinatoire.

Abstract:

An empirical study of two well known exact methods of solution of combinatorial optimization problems: branch and bound and dynamic programming. For the first we have considered the model that is applied to solve integer linear programming problems. As for the second we have considered the one that is applied to sequential problems. Here we have taken the scheduling problem of jobs on a machine to minimize the average tardiness.

Key-words: branch and bound, dynamic programming, scheduling, linear programming, combinatorics.

الملخص:

دراسة عددية أنجزت لطريقين لإيجاد حل امثل لمشاكل تحليل توفيقية:

طريقة التفريع والربط وطريقة البرمجة الديناميكية. بالنسبة للطريقة الأولى اعتبرنا النموذج المطبق على حل البرنامج الخطي ذو المتغيرات الصحيحة, بينما بالنسبة للطريقة الثانية فقد طبقناها على حل مشكل جدولة أشغال على ماكنة والهدف هو تصغير معدل التاخرات.

الكلمات المفتاحية: التفريع والربط , البرمجة الديناميكية, الجدولة, البرمجة الخطية, التحليل توفيقيي.