République Algérienne Démocratique et Populaire.
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique.
Université Mouloud Mammeri de Tizi-Ouzou

Faculté des Sciences
Département des Mathématiques

Mémoire de Master

Filière : MATHÉMATIQUES APPLIQUES
Spécialité :Recherche Opérationnelle

Approximation de quelques problèmes d'optimisation combinatoire

Réalisé par :

BOULTOUAK MELISSA
KABLI LYDIA

Devant le jury :

Mr D.TALEM	MCB	U.M.M.T.O	Président.
Mme R.KHEFFACHE	MCB	U.M.M.T.O	Promotrice.
Mme F.AKLOUCHE	MAA	U.M.M.T.O	Examinatrice.

Année universitaire :2022/2023

Remerciements

Le grand merci pour Allah, Grand et Miséricordieux pour le courage, la patience et la force qui nous a donné pour mener à bien ce travail.

Tout d'abord, nous tenons à remercier notre encadrant M^{me} R.KHEFFACHE MCB, d'avoir accepté de diriger ce travail, sa patience, sa disponibilité, son temps précieux qu'elle nous a accordé, aussi pour ses précieux conseils et son aide durant toute la période du travail. Sa compétence et son encadrement ont toujours suscité notre profond respect. On espère nous montrer digne de votre confiance. Nous nous exprimons notre sincère remerciement pour votre aide apporté à l'élaboration de ce mémoire.

Nos vifs remerciements vont également aux membres du jury Mr D.TALEM MCB et Mme F.AKLOUCHE MAA pour l'honneur qu'ils nous font en acceptant d'évaluer notre modeste travail. Nous saisissons cette occasion pour vous exprimer nos sentiments de respect et de gratitude.

Nous souhaitons exprimer notre gratitude envers nos familles et nos amis(es) qui nous ont soutenus tout au long de cette aventure. Leur amour, leur encouragement et leur compréhension ont été essentiels pour surmonter les moments de doute et de stress.

Que toute personne ayant participé de près ou de loin dans l'élaboration de ce travail, trouve ici l'expression de nos très vifs remerciements.

Dédicace

À nos très chers parents.

Aucune dédicace aussi parfaite et douce soit elle, notre reconnaissance, notre gratitude, notre respect le plus profond et tout l'amour qu'on vous porte.

Ce travail représente le fruit de votre soutien, vos sacrifices et vos encouragements.

À tous nos amis(es)
À tous ceux qui nous sont chers

TABLE DES MATIÈRES

TABLE DES MATIÈRES

INTRODUCTION GÉNÉRALE

L'optimisation combinatoire est une branche importante de l'informatique et de mathématique qui s'intéresse à la recherche de solutions optimales dans un domaine discret. Elle englobe de nombreux problèmes d'optimisation, tels que le problème du voyageur de commerce, le problème du sac à dos, l'affectation de ressources, la coloration de graphes, etc.
Ces problèmes sont souvent NP-difficiles, ce qui signifie qu'il est peu probable qu'il existe un algorithme efficace permettant de les résoudre exactement en temps polynomial.
Face à cette difficulté, les chercheurs se sont tournés vers les méthodes dites approchées.
Ces méthodes visent à résoudre des problèmes d'optimisation de très grande taille et donnent des solutions d'excellente qualité au prix d'une consommation en ressources réduite.
La perte du caractère optimal se voit donc compensée par la diminution du temps de calcul.
Une classe bien particulière de méthodes approchées est l'approximation polynomiale qui permet de fournir des algorithmes qui garantissent la qualité de leurs solution, c'est- à-dire capable de calculer des solutions dont la distance en valeur à la solution optimale est petite.
Cette qualité est souvent exprimée en fonction d'un facteur d'approximation.
Notre travail est divisé en quatre chapitres :

Dans ce premier chapitre, nous présenterons une collecte de définitions et de notions de base de la théorie des graphes utilisées dans notre travail.

Ensuite dans le deuxième chapitre, nous avons présenté les notions fondamentales de l'optimisation combinatoire, ainsi que la complexité et les classes de complexité qui sont es-

sentielles pour comprendre la difficulté algorithmique inhérente à ces problèmes.

Dans le troisième chapitre, nous avons abordé les notions de base de l'approximation polynomiale.

Enfin dans le quatrième chapitre, nous avons présenté les algorithmes d'approximations pour les problèmes du cycle hamiltonien, chaîne hamiltonniene à deux dépôts et le problème du sac à dos que nous avons illustré à l'aide d'exemples .
Puis, nous terminons ce mémoire par une conclusion et une bibliographie.

CHAPITRE 1

CONCEPTS DE BASE DE LA THÉORIE DES GRAPHES

1.1 Introduction :

La théorie des graphes est un outils puissant de modélisation et de résolution des problèmes concrets.
Dans ce chapitre, nous présentons quelques définitions et concepts de base de la théorie des graphe.

1.2 Vocabulaire de la théorie des graphes :

1.2.1 Graphe :

Un graphe est un schéma constitué d'un ensemble de points appelés sommets (noeuds) et d'un ensemble de lignes reliant ces sommets appelées arêtes ou arcs. Formellement un graphe est un couple G=(V,E) comprenant un ensemble de sommets V et un ensemble d'arêtes E.

Le graphe G est dit orienté ou non-orienté selon que la relation induite par E est respec-

tivement asymétrique ou symétrique.
Une illustration de ce concept est donnée par la figure 3.1. Dans le cas non-orienté, nous parlerons d'arêtes (Edges en anglais) et non d'arcs.

Le nombre de noeuds contenus dans un graphe G est appelé ordre de G. Par convention, il est noté n. De même, le nombre d'arcs (ou d'arêtes, selon l'orientation du graphe) contenues dans un graphe G est noté m.

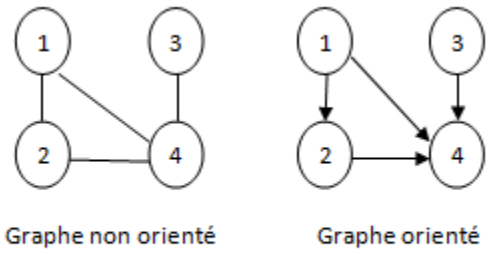


FIGURE 3.1 Exemples de graphes

1.2.2 Adjacence et voisin :

Deux sommets u et v sont adjacents dans G si u et v sont reliés par au moins une arête. Deux arêtes (arcs) sont adjacents si et seulement si elles (ils) ont une même extrémité.
Un voisin d'un sommet u est un sommet qui est adjacent à u.[4]

1.2.3 Degré d'un sommet :

Le degré d'un sommet u dans le graphe G noté d(u) est le nombre d'arêtes incidentes à u.

1.2.4 Cycle

Un cycle est une chaîne dont l'extrémité initiale se confond avec l'extrémité terminale.

Un cycle (resp chaîne) qui passe exactement une fois par chaque arête d'un graphe est dit eulérien (resp eulerienne).
Un cycle (resp chaîne) qui passe exactement une fois par chaque sommet d'un graphe est dit hamiltonien (resp Hamiltonienne).

1.2.5 Chaîne :

une chaîne est une séquence finie et alternée de sommets et d'arêtes, débutant et finissant par des sommets, telle que chaque arête est incidente avec les sommets qui l'encadre dans la séquence.

1.2.6 Boucle :

Une boucle est une arête reliant un sommet à lui-même.
Autrement dit, c'est une arête dont ses deux extrémités correspondent au même sommet du graphe.

Graphe simple :

Un graphe simple est un graphe sans boucle dans lequel deux sommets sont reliés par au plus une arête (ou arc).

Graphe complet :

Le graphe complet est un graphe simple à n sommets dont tous les sommets sont adjacents.

1.2 Vocabulaire de la théorie des graphes :

Graphe partiel :

soit G = (V, E) un graphe. Un graphe partiel, noté H = ($V^{'}$, $E^{'}$), est un sous-ensemble des sommets $V^{'} \subseteq V$ et un sous-ensemble d'arêtes $E^{'} \subseteq E$, tels que chaque arête dans $E^{'}$ connecte deux sommets qui appartiennent à $V^{'}$.

Graphe connexe :

Un graphe G est connexe si pour chaque paire de sommets (x,y) de G il existe une chaîne reliant x à y.

Sous Graphe :

On obtient le Sous Graphe G_s en enlevant un ou plusieurs sommets au graphe G, ainsi que toutes les arêtes incidentes aux sommets enlevés.

Graphe pondéré :

Un graphe pondéré G=(V,E,W) est un graphe où un entier positif est affecté à chaque arête.
On appelle cet entier poids de l'arête.

Poids d'un graphe :

Le poids ou coût d'un graphe est la somme des poids des arêtes du graphe on le note W(G).

1.2.7 Arbre :

Un arbre est un graphe connexe et sans cycle. Il satisfait l'égalité suivante : |E| = |V| - 1.

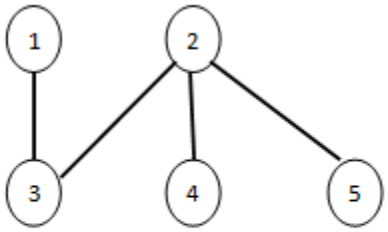


FIGURE 3.2 Un arbre

Théorème 1.1. Dans un arbre le nombre de sommets de degré impair est toujours pair.

1.2.8 arbre couvrant :

Un arbre couvrant T d'un graphe G=(V,E) est un graphe partiel sans cycle.

Proposition 1.1. Un graphe admet un arbre couvrant si, et seulement si, il est connexe.

1.2.9 Arbre couvrant de poids minimum :

Soit un graphe G=(V,E,W) un graphe non orienté pondéré
On appelle arbre couvrant minimum (maximum) : Tout arbre couvrant dont la somme des poids des arêtes le constituant est minimal (maximal).

Pour trouver un arbre de coût minimum dans un graphe, on utilise l'algorithme de Kruskal :

1.2 Vocabulaire de la théorie des graphes :

Algorithm 1 Algorithme de Kruskal [1]

G=(V,E), |V|=n, |E|=m
On ordonne les arêtes de G par rapport à leur coût dans l'ordre croissant :

$$C(e_1) \leq C(e_2) \leq ... \leq C(e_m)$$

Soit T=(V,Y) l'arbre recherché i=1, Y=$\emptyset$
(1)*Si* (V,Y$\cup\{e_i\}$) contient un cycle, aller en (3)
Si $(V, Y \cup \{e_i\})$ ne contient pas de cycle, aller en (2)
(2) $Y = Y \cup \{e_i\}$) aller en (3)
(3) Si i=m alors terminer
sinon i := i+1 ; aller en (1).

Exemple 1.1. Soit un graphe :

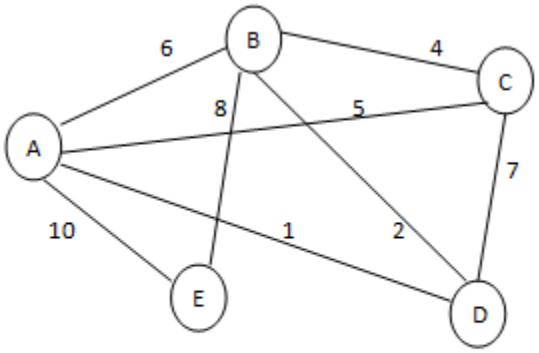


FIGURE 3.3

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
Arêtes	(A,D)	(B,D)	(B,C)	(A,C)	(A,B)	(D,C)	(B,E)	(A,E)
Coût	1	2	4	5	6	7	8	10

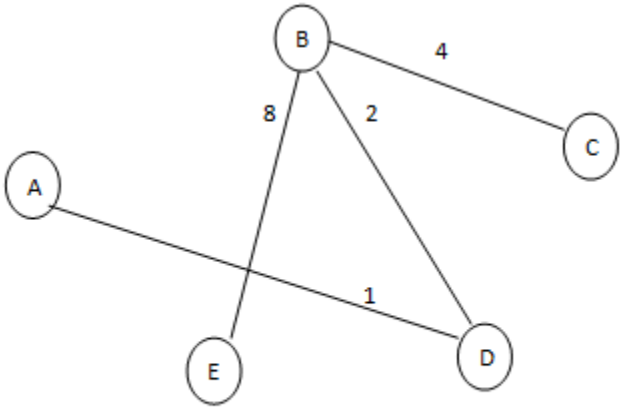


FIGURE 3.4 : Arbre de coût minimum

C(T)= 1+2+4+5+8+10 = 30

1.2.10 Forêt :

Une forêt est un ensemble de graphes non orientés (ou d'arbres) qui sont disjoints les uns des autres. En d'autres termes, une forêt est un ensemble de sous-graphes non orientés, où chaque sous-graphe est un arbre (un graphe acyclique connexe)

1.2.11 Chemin :

Un chemin est une séquence finie et alternée de sommets et d'arcs , débutant et finissant pas des sommets , telle que chaque arc est sortant d'un sommet est incident au sommet suivant dans les séquence.

1.2.12 Couplage :

Soit un graphe simple non orienté, un couplage M est un ensemble d'arêtes deux à deux non adjacentes.

1.2 Vocabulaire de la théorie des graphes :

Couplage parfait :

Un sommet v est saturé par un couplage M si v est l'extrémité d'une arête de M. Dans le cas contraire, v est insaturé.
Un couplage parfait est un couplage où chaque sommet du graphe est saturé.
Un couplage de poids minimum dans un graphe est un couplage où la somme des poids des arêtes sélectionnées est minimale parmi tous les couplages parfaits du graphe.

Algorithme de recherche d'un couplage parfait de coût minimum :

Algorithm 2 Algorithme de recherche d'un couplage parfait : [10]

1- Initialisation : Commencez avec un couplage de M
2- Tant qu'ils existent des noeuds non couplés :
Sélectionner un noeud non couplé et trouvé l'arête de coût minimum reliant ce noeud à un noeud non couplé. Ajouter cette arête au couplage.
3- Retourner le couplage obtenu.

Exemple 1.2. Appliquant cet algorithme pour l'exemple suivant :

	a	b	c	d
A	10*	8	7	12
B	11	7	6*	10
C	9	5*	8	9
D	12	6	7	6*

On obtient le couplage parfait suivant :

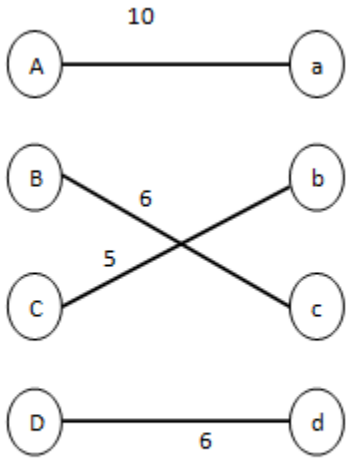


FIGURE 3.5

C(M)=10+6+6+5=27

CHAPITRE 2

OPTIMISATION COMBINATOIRE ET COMPLEXITÉ

2.1 Introduction

Ce chapitre englobe quelques notions élémentaires de la théorie de l'optimisation. Dans un premier temps, nous rappelons les définitions et concepts de base de l'optimisation combinatoire, et la notion de complexité, qui sont nécessaires pour la compréhension de notre travail.

2.2 Optimisation Combinatoire :

L'optimisation combinatoire est une discipline qui combine divers techniques des mathématiques discrète et de l'informatique afin de résoudre des problèmes de la vie réelle.
Son domaine est celui des problèmes d'optimisation où l'ensemble des solutions réalisables est discret. Elle consiste à trouver la meilleure solution parmi un ensemble fini (mais souvent très grand) de possibilités.

De nombreuses applications peuvent être formulées sous la forme d'un problème d'opti-

misation combinatoire qui est défini par la donnée d'un ensemble S et d'une application f : S $\rightarrow \mathbb{R}$, il s'agit de déterminer s^* tel que $f(s^*) \geq$ f(s) pour tout élément de S (problème de maximisation) ou $f(s^*) \leq$ f(s) pour tout élément de S (problème de minimisation).

2.2.1 Problème :

Dans la théorie algorithmique, un problème est une question générale pour laquelle on veut obtenir une réponse. Cette question possède généralement des paramètres ou des variables dont la valeur reste à fixer.
Un problème est spécifié en donnant la liste de ces paramètres ainsi que les propriétés que doit vérifier la réponse.

Instance :

Une instance d'un problème est obtenue en affectant une valeur à tous les paramètres de ce problème.

2.3 Complexité :

Se réfère généralement à la difficulté de résoudre un problème en fonction de la taille de l'instance du problème, la taille de l'instance peut être mesurée de différentes manières selon le problème spécifique considéré, mais elle est généralement liée aux nombre d'éléments ou de variables dans l'instance.

La théorie de la complexité a été développée pour classer les problèmes faciles et difficiles.

2.3.1 Algorithme :

Un algorithme est une suite d'opérations élémentaires (affectations de variables, tests, etc.) qui, lorsqu'on lui fourni une instance d'un problème en entrée, s'arrête après exécution de la dernière opération en nous renvoyant la solution.

2.3.2 Complexité d'un algorithme :

Les deux paramètres les plus importants pour mesurer la qualité d'un algorithme sont : son temps d'exécution et l'espace mémoire qu'il utilise.

Le premier paramètre s'exprime en termes de nombre d'instructions nécessaires au déroulement de l'algorithme. L'utilisation du nombre d'instructions comme unité de temps est justifiée par le fait qu'un même programme utilisera le même nombre d'instructions sur deux machines différentes mais prendra plus ou moins de temps selon leurs rapidités respectives. On considère généralement qu'une instruction correspond à une opération élémentaire, par exemple, une affectation, un test, une addition, une multiplication, un marquage , etc. Ce que l'on appelle la complexité en temps ou simplement la complexité d'un algorithme correspond à peu près à une indication du temps qu'il prendra pour résoudre un problème d'une taille donnée. C'est en réalité une fonction qui associe à la taille d'une instance d'un problème donné, un ordre de grandeur du nombre d'instructions nécessaire à sa résolution. Le deuxième paramètre correspond au nombre de cases mémoire utilisées par l'algorithme afin de résoudre un problème.

2.3.3 Complexité dans le pire cas :

Il y a plusieurs ensembles d'hypothèses quant à la configuration type qui nous sert de base pour mesurer la complexité d'un algorithme. Le cadre de travail le plus souvent utilisé est le cadre dit du pire des cas.

Ici, la complexité d'un algorithme est le nombres d'opérations effectuées sur l'instance qui représente la pire configuration, parmi celles d'une taille fixée, pour son déroulement.

2.3.4 Algorithme efficace :

Un algorithme est dit efficace s'il est capable de résoudre un problème en un temps raisonnable, même pour des entrées de taille importantes. C'est à dire sa complexité est majorée par un polynôme en la taille des données.

2.3.5 Complexité d'un problème :

La définition de la complexité d'un algorithme peut facilement se transporter sur les problèmes. Informellement la complexité d'un problème est la complexité du meilleur algorithme qui le résout.

2.3.6 Classes des problèmes :

Pour pouvoir exposer la notion de classes des problèmes, il est tout d'abord nécessaire de distinguer les problèmes de décision des problèmes d'optimisation combinatoire.

Un problème de décision est un problème dont la solution est Oui ou Non. Il est possible d'associer à chaque problème d'optimisation un problème de décision en introduisant un seuil α correspondant à la fonction f. Le problème de décision devient : existe-t-il une solution réalisable s telle que f(s)$\leq$ ou f(s) $\geq \alpha$?.

Classe des problèmes P :

Un problème est de classe P s'il existe un algorithme polynomial pour le résoudre. Les problèmes appartenant à cette classe sont dits faciles.

Classe des problèmes NP :

Un problème de décision appartient à la classe NP si étant donnée une proposition d'une solution au problème , pour lequel on peut vérifier en un temps polynomial si une solution proposée convient.[8]

Classe des problèmes NP-complet :

La classe NP complet regroupe les problèmes les plus difficiles de la classe NP .
Si nous sommes capables de trouver un algorithme pour résoudre efficacement un problème de NP-complet alors, nous pouvons utiliser cet algorithme pour résoudre tous les problèmes de NP.

Classe des problèmes NPO :

La classe NPO est la classe des problèmes d'optimisation dont la variante décisionnelle est dans NP.

2.4 Conclusion

Dans ce chapitre, nous avons présenté quelques notions de base sur l'optimisation combinatoire et la notion de complexité des problèmes.

CHAPITRE 3

MÉTHODES DE RÉSOLUTION

3.1 Introduction

Dans ce chapitre, nous allons présenter de manière générale les méthodes de résolution des problèmes d'optimisation. Ces méthodes peuvent être classées en deux grandes catégories : méthodes exactes et méthodes approchées.

3.2 Méthodes exactes :

Les méthodes exactes sont des techniques de résolution en optimisation qui garantissent la solution optimale à un problème donné. Elles permettent aussi de fournir des informations sur les bornes inférieures ou supérieures de la solution optimale dans le cas où l'algorithme est arrêté avant de terminer la résolution du problème.
Les méthodes exactes[7] ont l'avantage de fournir des solutions optimales, mais leurs complexités peuvent être très élevées pour des problèmes de grande taille.
De façon pratique seuls les problèmes de petite ou moyenne taille peuvent être résolus de façon optimale par des algorithmes exacts.

De plus pour certains problèmes, la consommation de mémoire de ces algorithmes exacts peut être très grand et peut parfois entraîner l'arrêt prématuré de l'application. L'une des méthodes exactes la plus utilisée, la méthode de séparation et évaluation dite (Branch and Bound) où la séparation permet d'obtenir une méthode générique pour énumérer toutes les solutions tandis que l'évaluation évite l'énumération systématique de toutes les solutions.

On distingue aussi la méthode de la programmation linéaire utilisée pour résoudre les problèmes d'optimisation où les fonctions objectifs et les contraintes sont linéaires. Tandis que la programmation en nombres entiers est utilisée pour résoudre des problèmes où les variables de décision sont discrètes.

Cependant les méthodes exactes peuvent être très coûteuses en temps et en ressources pour des problèmes de grande taille. C'est la raison pour laquelle de nombreuses méthodes approchées ont été adaptées à la résolution de ces problèmes.

3.3 Méthodes approchées

Pour les problèmes de grande taille, les méthodes exactes ne sont pas envisageables de par leur temps de calcul qui est exponentiel et donc sans aucun intérêt en pratique. Dans ce cas, il est possible d'utiliser des méthodes approchées qui fournissent une solution réalisable aussi bonne que possible. Ce sont souvent des méthodes heuristiques.
Une heuristique est une méthode ou technique de calcul approchée qui fournit une solution admissible, non nécessairement exacte, dans un temps polynomial pour un problème spécifique.

Ces méthodes sont souvent très satisfaisantes dans le cas où l'obtention d'une solution optimale n'est pas absolument requise.

Il est naturel de s'intéresser alors à la qualité des solutions fournies par l'algorithme utilisé et avoir une garantie quant à cette qualité. C'est le rôle de l'approximation polynomiale.

3.4 Approximation polynomiale

L'approximation polynomiale est apparue après le théorème de Cook [6, 5] initiant le développement de la complexité.
Elle consiste à construire des algorithmes qui donnent des solutions réalisables en un temps polynomial pour les problèmes NP-difficiles. Ces solutions garantissent une certaine qualité qu'on mesure avec un rapport dit : rapport d'approximation.

3.5 Rapport d'approximation :

Comme mentionné ci-dessus, puisque le but de l'approximation polynomiale est d'assurer et de fournir une solution ayant une garantie de bonne qualité, c'est-à-dire de définir une mesure de la qualité d'une solution.
L'estimation de la qualité d'un algorithme est faite à l'aide de rapport d'approximation, qui a pour rôle de mesurer la distance de la valeur de la solution par rapport à la valeur optimale.

3.5.1 Rapport d'approximation standard :

Le rapport standard consiste à comparer la valeur de la solution donnée par l'algorithme à la valeur optimale. Il s'agit bien d'une estimation de ce rapport, puisque l'optimum n'est pas connu à priori.
Soit A un algorithme polynomial pour le problème (P) et S une solution réalisable d'une instance I du problème (P), et "opt" la valeur d'une solution optimale de I. Le rapport d'approximation standard de S sur I est :

$$\rho(I) = \frac{S(I)}{Opt(I)}.$$

Ce rapport est dans [0,1] si (P) est à maximiser et dans $[1, +\infty]$ si (P) est à minimiser. Plus le rapport est proche de 1, meilleure est la solution.

3.5.2 Rapport d'approximation différentiel :

Le rapport différentiel a été introduit en 1977 [3]. Dans ce rapport, on mesure la qualité d'une solution en fonction de la valeur d'une pire solution notée $\lambda(I)$ qui est la valeur d'une solution optimale du problème obtenu en changeant le but du problème (maximiser devient minimiser et inversement) .
Le rapport d'approximation différentiel de S sur I est :

$$\delta(I) = \frac{\lambda(I) - S(I)}{Opt(I) - \lambda(I)}.$$

Par convention, lorsque $Opt(I) = \lambda(I)$ on fixera $\delta(I) = 1$ pour toute solution S(I).
Plus on se rapproche de l'optimum, plus le rapport se rapproche de 1.

3.6 Classes d'approximation :

Il est très vite apparu dans l'étude de l'approximation polynomiale que pour certains problèmes, on obtient des algorithmes à rapport constant pour d'autres la tâche semble plus difficile. L'idée de les regrouper dans des classes reflétant un niveau d'approximation est venue pour différencier les problèmes selon leurs propriétés d'approximation.
Une première classe est constituée des problèmes approximables à rapport constant (c'est-à-dire indépendant de l'instance).
Un problème (P) de NPO est dans APX(respectivement dans DAPX) s'il existe une constante

telle que (P) est approximable à rapport standard (respectivement différentiel).

APX (Approximabilité polynomiale) :

La classe APX regroupe les problèmes d'optimisation combinatoire pour lesquels il existe des algorithmes d'approximation polynomial avec une garantie de performance. Plus précisément, pour les problèmes appartenant à APX, il existe un algorithme polynomial qui, pour toute instance du problème, fournit une solution dont la valeur est au moins un facteur constant de la valeur de la solution optimale.

Exemple 3.1. Un exemple de problème dans la classe APX est le problème du voyageur de commerce (TSP). Il existe des algorithmes d'approximation polynomiaux, tels que l'algorithme de Christofides, qui garantissent une solution dont la valeur est au plus 1,5 fois la valeur de la solution optimale.

DAPX (Différentiellement approximable) :

La classe DAPX regroupe les problèmes d'optimisation combinatoire pour lesquels il est possible de concevoir des algorithmes d'approximation qui sont différenciellement approximables par rapport à une solution de référence donnée. Cela signifie que pour tout problème appartenant à DAPX, il existe un algorithme polynomial qui fournit une solution dont le rapport d'approximation différentiel est borné par un facteur constant par rapport à la solution de référence.

Exemple 3.2. Un exemple de problème dans la classe DAPX est le problème de couverture par sommets (Vertex Cover). Pour toute solution de référence donnée, il est possible de concevoir un algorithme d'approximation polynomial qui fournit une solution dont le rapport d'approximation différentiel est borné par un facteur constant par rapport à la solution de référence.

On distingue aussi d'autres problèmes pour lesquels, on ne peut trouver un algorithme approché à rapport constant. Dans ce cas, on peut approcher ces problèmes par une fonction f qui dépend de la taille de l'instance I, qui tendra vers l'infini dans le cas d'une minimisation en approximation standard et vers zéro dans les autres cas. Il s'agit de trouver une meilleure fonction f celle qui tend le moins vite vers l'infini ou zéro.

Les deux classes de fonctions les plus classiques sont les fonctions polynomiales et algorithmiques.

Il existe d'autres classes d'approximation que nous n'avons pas évoquées dans ce chapitre, vu que notre travail n'a pas de rapport avec ces classes.

3.7 Conclusion

Nous avons présenté les notions préliminaires sur l'approximation polynomiale.

CHAPITRE 4

APPROXIMATION DE QUELQUES PROBLÈMES D'OPTIMISATION

4.1 Introduction :

Dans ce chapitre, nous présentons des algorithmes d'approximation pour trois types de problèmes d'optimisation.

Le premier est le problème de cycle hamiltonienne , le second est le problème de la chaîne hamiltonienne à deux dépôts et enfin, le problème du sac à dos.

Ces problèmes sont très étudiés en optimisation combinatoire en effet leurs applications sont nombreuses :
Dans le domaine de la logistique, le cycle hamiltonien est utilisé pour optimiser les itinéraires de livraison, en s'assurant que chaque lieu est visité une seule fois sans gaspillage de temps ou de ressources.
Dans les réseaux de communication, la recherche d'une chaîne hamiltonienne est essentielle

pour établir des itinéraires de transmission de données sans interruption, améliorant ainsi l'efficacité et la fiabilité des communications.
Dans le domaine de la gestion des stocks, le problème du sac à dos est appliqué pour déterminer quels articles doivent être placés dans l'inventaire en fonction de leur valeur et de leur poids, optimisant ainsi l'utilisation de l'espace et des ressources.

4.2 Problème du cycle hamiltonien

Le problème du cycle hamiltonien en optimisation combinatoire est un problème classique qui consiste à trouver un cycle qui passe par tous les sommets d'un graphe exactement une fois, à l'exception du sommet de départ qui est également le sommet d'arrivée.
Ce problème est connu sous le nom du problème de voyageur de commerce. Il a été posé pour la première fois par William Rowan en 1859 [11].

Formellement, étant donné un graphe non orienté G = (V, E) où V représente l'ensemble des sommets et E l'ensemble des arêtes, le problème du cycle hamiltonien consiste à déterminer s'il existe un cycle qui visite chaque sommet de V exactement une fois et revient au sommet de départ.

Ce problème est connu pour être NP-complet, ce qui signifie qu'il n'existe pas d'algorithme polynomial pour le résoudre .

Les domaines d'application sont nombreuses : problème de logistique, de transport...

4.3 Approximation du problème du cycle hamiltonien :

Soit G = (V,E) un graphe complet, on affecte à chaque arête (i,j) un coût $C_{ij} = C_{ji}$ pour chaque i,j $\in V$ tel que l'inégalité triangulaire soit vérifiée, c'est-à-dire :

$$C(i,j) \leq C(i,k) + C(k,j), \forall i,j,k \in V \ .$$

Le problème est de trouver un cycle hamiltonien de coût minimum. Pour cela, nous avons utilisé l'algorithme de Christofides suivant :

4.3 Approximation du problème du cycle hamiltonien :

Algorithm 3 Algorithme de Christofides [2] :

Étape 1 : Calculer l'arbre couvrant T du poids minimum à l'aide de l'algorithme de Kruskal.
Étape 2 : Trouver les sommets de degré impair dans l'arbre T.
Étape 3 : Chercher un couplage parfait M à l'aide des sommets de degré impair.
Étape 4 : Ajouter les arêtes du couplage parfait à l'arbre pour obtenir un tour eulérien c'est-à-dire chacun des sommets est de degré pair.
Étape 5 : Appliquer le shortcuting pour avoir un tour hamiltonien : Pour tout sommet v de degré > 2, considérer deux arêtes (u,v) et (v,w) consécutives dans le tour eulérien et les remplacer par l'arête (u,w).

Théorème 4.1. L'algorithme présente une approximation de rapport de $\frac{3}{2}$ pour le problème du cycle hamiltonien.

Preuve. Soit Cy le cycle obtenu par l'algorithme et C^* le cycle hamiltonien optimal
Montrons que $C(Cy) \leq \frac{3}{2}C(C^*)$
on a :
Le coût du cycle eulérien trouvé par l'algorithme égale à la somme des coûts de l'arbre T et le coût du couplage M
et on a aussi :

$$C(Cy) \leq C(T) + C(M)$$

d'un autre coté, on a :

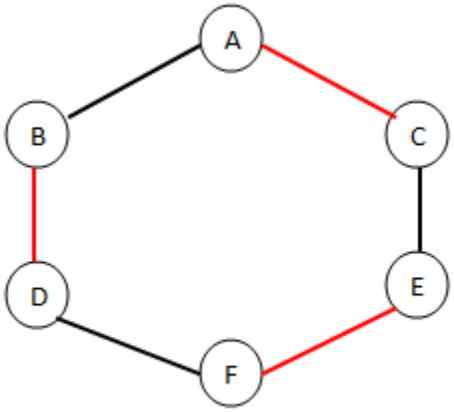
$$C(T) \leq C(C^*).....(1)$$

Car l'arbre T est obtenu en enlevant une arête d'un cycle .
Montrons maintenant que : $C(M) \leq \frac{1}{2}C(C^*)$
Soit $C^{'}$ le cycle obtenu de C^* en supprimant les sommets hors V_0.
L'inégalité triangulaire donne :

4.3 Approximation du problème du cycle hamiltonien :

$$C(C') \leq C(C^*)$$

C' est un cycle contenant uniquement les sommets de V_0
Comme $|V_0|$ est pair, C' est l'union de deux couplages parfait sur V_0.



L'un des deux couplages est de coût$\leq \frac{C(C')}{2} \leq \frac{C(C^*)}{2}$
Comme M est un couplage parfait de coût minimum sur V_0, on a :

$$C(M) \leq \frac{C(C^*)}{2} .....(2)$$

De (1) et (2) on a :

$$C(Cy) \leq C(T) + C(M)$$
$$C(T) \leq C(C^*).....(a)$$
$$C(M) \leq \tfrac{1}{2}C(C^*).....(b)$$

4.3 Approximation du problème du cycle hamiltonien :

(a) et (b) $\Rightarrow C(Cy) \leq C(T) + C(M) \leq (1 + \frac{1}{2})C(C^*) \leq \frac{3}{2}C(C^*)$

d'où :

$$C(Cy) \leq \frac{3}{2}C(C^*).$$

Exemple 4.1. Soit G=(V,E) un graphe complet à 12 sommets, tels que les coûts vérifient l'inégalité triangulaire.

1- Soit un arbre de coût minimum :

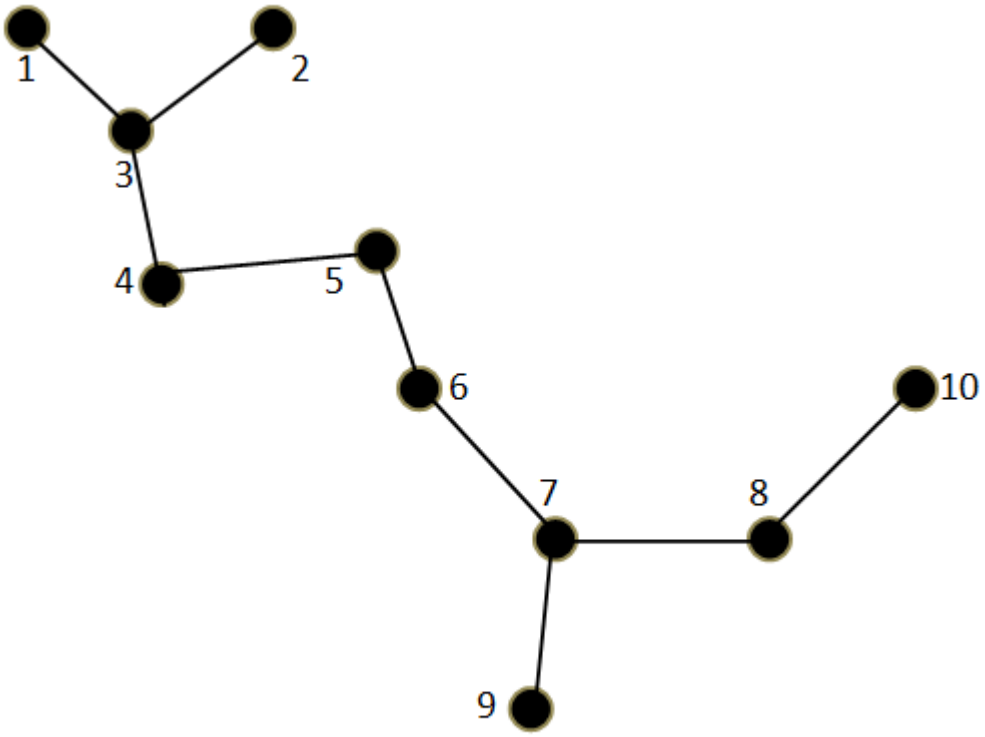


FIGURE 4.1

4.3 Approximation du problème du cycle hamiltonien :

2- Trouvons un couplage parfait de coût minimum à l'aide de tous les sommets de degré impair de l'arbre couvrant minimum :

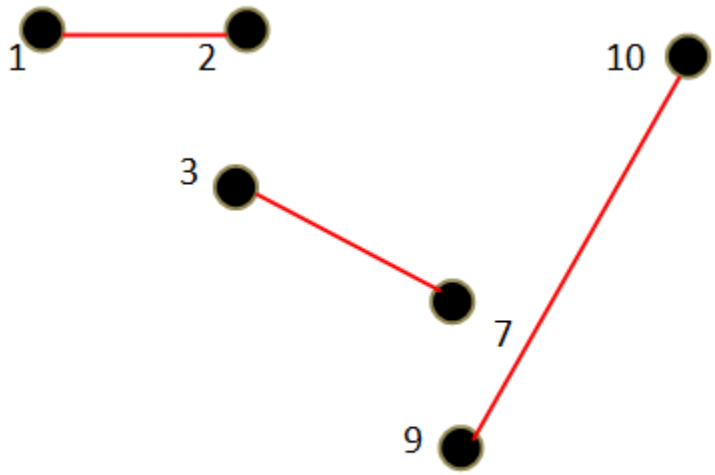


FIGURE 4.2

3- Ajoutons toutes les arêtes du couplage parfait à l'arbre pour obtenir un graphe eulérien
.

4.3 Approximation du problème du cycle hamiltonien :

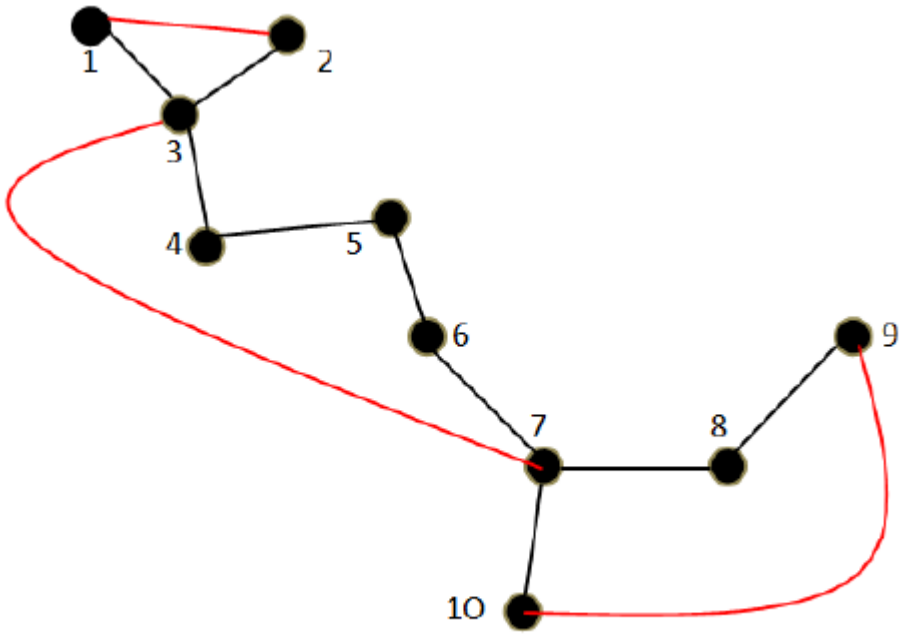


FIGURE 4.3

4- Raccourcir le tour eulérien pour obtenir un cycle hamiltonien, on appliquant le short-cuting :

(10 7 3 1 2 ~~3~~ 4 5 6 ~~7~~ 8 9 10)

4.3 Approximation du problème du cycle hamiltonien :

Ainsi on obtient le cycle hamiltonien suivant :

(10 7 3 1 2 4 5 6 8 9 10)

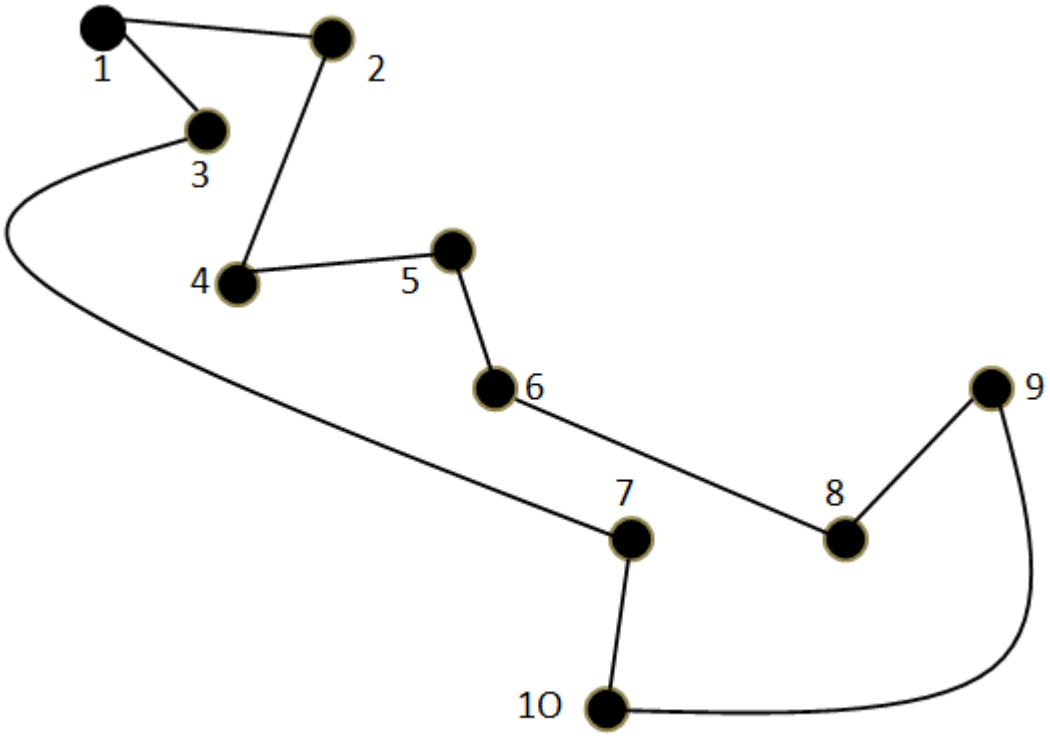


FIGURE 4.4

4.4 Approximation du problème de chaîne hamiltonienne à deux dépôts

4.4.1 Définition du problème :

Soit $D = \{d_1, d_2\}$ un ensemble de deux dépôts distincts $U = \{1, 2, 3, ...n\}$ un ensemble de n sommets avec $(n \geq 2)$ et $V = D \cup U$. On affecte à chaque *arête* (i,j) joignant les sommets i et j , un coût $C_{ij} \in \mathbb{Q}$ avec $C_{ij} = C_{ji}$ pour tout $i, j \in V$ et $C_{ik} \leq C_{ij} + C_{ik}$ pour tout $i, j, k \in V$(inégalité triangulaire). Une chaîne parcourue par un voyageur l est une séquence de sommets $P_l = \left\{d_l, v_1^l\ , v_2^l.....v_{k_l}^l\right\}$ où l=1,2 et k_l représente le nombre de sommets visités par le $l^{éme}$ voyageur et $\forall j \in \{1, 2, ...k_l\}$, $v_j^l \in U$.
Le coût de chaque chaîne P_l parcourue par le $l^{éme}$ voyageur est défini par :

$$C(P_l) = \begin{cases} C(d_l, v_1^l) + \sum_{j=1}^{k_l-1} C(v_j^l, v_{j+1}^l) & \text{si } k_l > 0, avec\ l = 1, 2. \\ 0 & \text{sinon.} \end{cases}$$

L'objectif du problème est de trouver les chaîne P_1 et P_2 tel que :
1.Chaque destination de U soit visitée une seule fois par un seul voyageur.
2.Chaque voyageur visite au moins une destination.
3.La somme totale des $co\hat{u}ts \sum_{l=1}^{2} C(P_l)$ soit minimum.
On note ce problème par 2DHPP (Two Depots Hamiltonian Path Problem).

4.4 Approximation du problème de chaîne hamiltonienne à deux dépôts

Algorithm 4 Chaîne hamiltonienne à deux dépôts [9]

1- Relier les deux dépôts à un sommet w dit racine par deux arêtes du coût zéro puis supprimer toutes les arêtes reliant ces dépôts.
Chercher un arbre du coût minimum dont la racine est w et supprimer les arêtes du coût zéro, ainsi nous obtenons une forêt F contenant deux arbres $T(d_1)$ et $T(d_2)$ tel que :

(a) Chaque arbre contient un seul dépôt.
(b) Chaque dépôt est de degré 1.
On note l'ensemble d'arêtes de la forêt par E(F) et les destinations de degré impair dans chaque arbre de F par O_i pour i=1,2.
2- Considérer le graphe $(V_0, E(V_0))$ où V_0 est l'ensemble des sommets de degré impair de la forêt F et $E(V_0)$ ensemble d'arêtes joignant deux sommets de V_0. Comme $|O_1|$ et $|O_2|$ sont impairs alors :

$$|V_o| = |O_1| + |O_2| \text{ est pair.}$$

Trouver un couplage partiel M contenant un sous ensemble d'arêtes $E(M) \subseteq E(V_0)$ de cardinal $\frac{|V_0|-2}{2}$ tel que le coût du couplage partiel soit minimum. On note les deux destinations non reliées par m_1 et m_2.
3- Ajouter les arêtes du couplage trouvées dans l'étape (2) à la forêt et relier les dépôts d_1 et d_2 par une arête du coût 0.
On considère le nouveau graphe

$$G_n = (V, E(F) \cup E(M) \cup \{(d_1, d_2)\})$$

Le degré de chaque sommet de G_n est pair sauf les sommets m_1 et m_2.
4- Trouver une chaîne eulérienne E_r dans le graphe G_n qui commence par m_1 et se termine par m_2 et visite chaque arête de G_n une seule fois.
5- Appliquer le shortcutting à la chaîne eulérienne E_r pour avoir une chaîne hamiltonienne où chaque sommet est visité une seule fois, de la manière suivante : On note cette chaîne par :

$$E_{sc} = (m_1, ..., d_1, d_2, .., m_2)$$

Supprimer les arêtes du coût zéro dans E_{sc} pour obtenir les deux chaînes hamiltoniennes parcourues par les deux voyageurs de commerce.

4.4 Approximation du problème de chaîne hamiltonienne à deux dépôts

Théorème 4.2. L'algorithme présente une approximation de rapport de $\frac{3}{2}$ pour le problème de la chaîne hamiltonienne à 2 dépôts (2DHPP).

Preuve. Montrons que l'approximation de cet algorithme est de $\frac{3}{2}$
Cela revient à montrer que : $C(S) \leq \frac{3}{2}C(Opt)$ avec C(S) est le coût de la solution produite par l'algorithme et C(Opt) le coût de la solution optimale.
D'un côté, on a la solution S est majorée par le coût de la forêt F noté par C(F) et le coût du couplage M noté par C(M) car l'algorithme détermine une forêt et un couplage de coût minimum, puis raccourcir les chaînes si elles passent plusieurs fois par un même sommet, donc : $C(S) \leq C(F) + C(M)$.
D'un autre côté, on a le $C(F) \leq C(Opt)$. Il suffit de montrer que le coût du couplage M est majoré par le coût de la solution optimale.
On considère une solution optimale (Opt) de 2DHPP, cette solution contient deux chaînes disjointes P_1^* et P_2^* parcourues par les deux voyageurs de commerce 1,2 respectivement.
On définit $H_l = \{v, v \in V_0, v \in P_l^*\}$ pour l=1,2. Chaque sommet de H_l est un sommet visité par le $l^{ème}$ voyageur de commerce dans la solution optimale est aussi un sommet de degré impair dans la forêt de coût minimum obtenue par l'algorithme.
On a $H_1 \cap H_2 = \emptyset$ avec $H_1 \cup H_2 = V_0$ et $|H_1| + |H_2| = |V_0|$ est pair.
Raccourcir les arêtes de la solution optimale telle que les chaînes P_1 et P_2 soient incidentes aux sommets de H_l pour l=1,2.

Les chaînes qui relient les sommets de H_1 et H_2 dans l'ordre de leur occurrence à d_1 et d_2 respectivement sont de coût inférieur ou égal au coût de la solution optimale et elles peuvent être décomposées en deux couplages partiels disjoints.
On distingue les cas suivants :

$1^{er} cas$:|H_1| et |H_2| pair

Soit $P_1 = \{u_1, u_2....u_{2p}\}$

4.4 Approximation du problème de chaîne hamiltonienne à deux dépôts

$P_2 = \{v_1, v_2 .... v_{2k}\}$
Dans ce cas, on considère la décomposition suivante :
$\mathbb{E}_1 = \{(u_1, u_2), (u_3, u_4), ..., (u_{2p-1}, u_{2p})\} \cup \{(v_2, v_3), (v_4, v_5), ..., (v_{2k-2}, v_{2k-1})\}$
$\mathbb{E}_2 = \{(u_2, u_3), (u_4, u_5), ..., (u_{2p-2}, u_{2p-1})\} \cup \{(v_1, v_2), (v_3, v_4), ..., (v_{2k-1}, v_{2k})\}$.

$\mathbb{E}_1$ et $\mathbb{E}_2$ sont deux couplages partiels disjoints.
Les sommets v_1 et v_{2k} ne sont pas adjacents à $\mathbb{E}_1$ et u_1 et u_{2p} ne sont pas adjacents à $\mathbb{E}_2$.

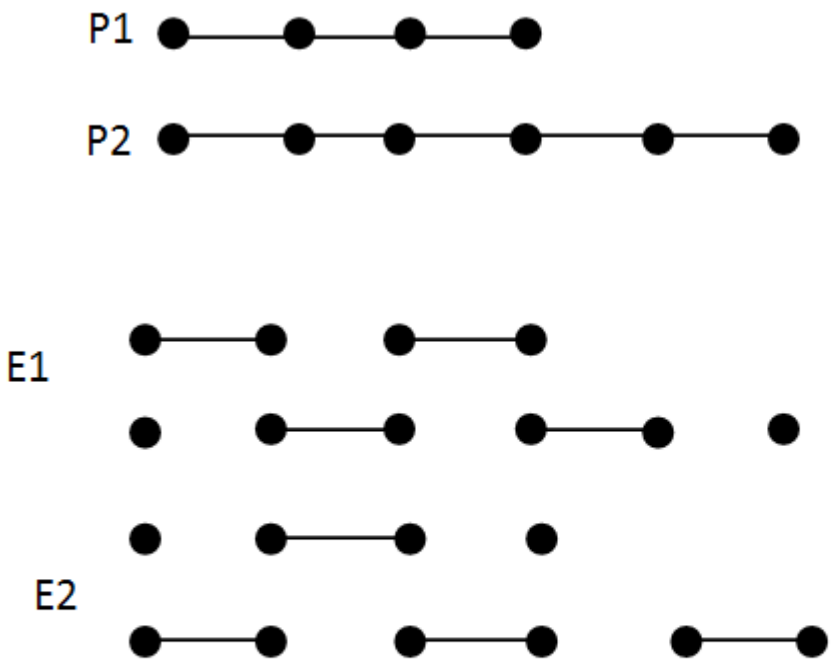


FIG : Décomposition des sommets de |Vo| en deux couplages disjoints

4.4 Approximation du problème de chaîne hamiltonienne à deux dépôts

$\mathbb{E}_1$ et $\mathbb{E}_2$ sont deux couplages disjoints de cardinal $\frac{|V_0|-2}{2}$, chacun de coût inférieur au coût du couplage partiel M.

$2^{\text{ème}} Cas$: | H_1| et $|H_2|$ impair
Soient $P_1 = \{u_1, u_2, u_3, ..., u_{2p+1}\}$

$P_2 = \{v_1, v_2, v_3.....v_{2k+1}\}$

Dans ce cas, on considère la décomposition suivante :

$\mathbb{E}_1 = \{(u_1, u_2), (u_3, u_4).....(u_{2p-1}, u_{2p})\} \cup \{(v_2, v_3), (v_4, v_5).....(v_{2k}, v_{2k+1})\}$
$\mathbb{E}_2 = \{(u_2, u_3), (u_4, u_5).....(u_{2p}, u_{2p+1})\} \cup \{(v_1, v_2), (v_3, v_4).....(v_{2k-1}, v_{2k})\}$

Les sommets u_{2p+1} et v_1 ne sont pas adjacents à $\mathbb{E}_1$
Les sommets u_1 et v_{2k+1} ne sont pas adjacents à $\mathbb{E}_2$

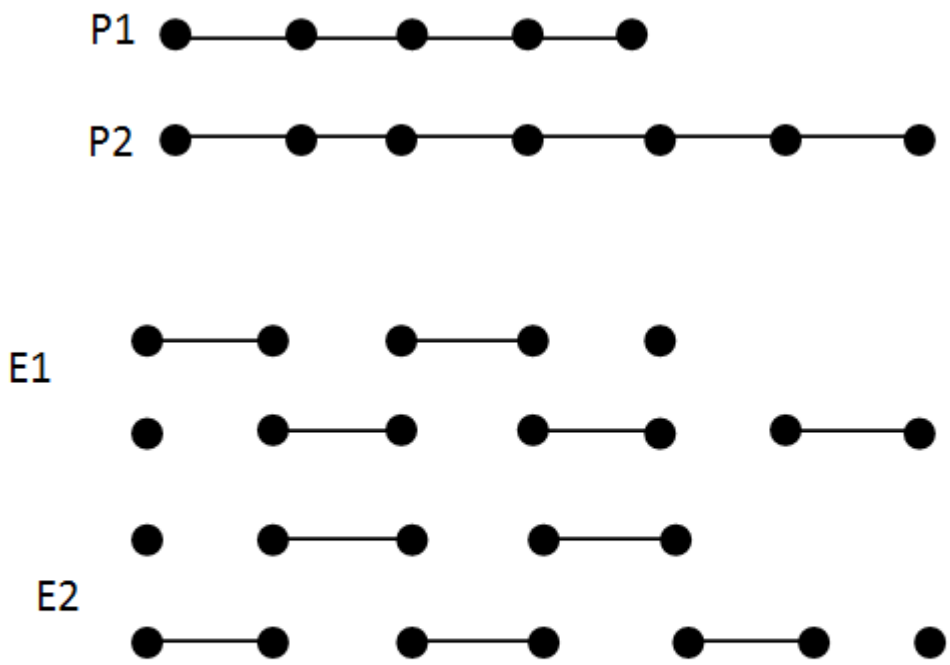


FIG : Décomposition des sommets de |Vo| en deux couplages disjoints

$\mathbb{E}_1$ et $\mathbb{E}_2$ sont deux couplages disjoints de cardinalité $\frac{|V_0|-2}{2}$ chacun de coût inférieur au coût du couplage partiel M.
d'après ce qui précède :

$$2C(M) \leq C(\mathbb{E}_1) + C(\mathbb{E}_2) \leq C(Opt)$$
$$C(M) \leq \tfrac{1}{2}C(Opt)$$

En résumé :

$$C(S) \leq C(M) + C(T)$$
$$C(M) \leq \frac{1}{2}C(Opt) \text{ .....(1)}$$
$$C(T) \leq C(Opt) \text{ .....(2)}$$

De (1) et (2) on obtient :

$$C(M) + C(T) \leq (\frac{1}{2} + 1)C(Opt) \leq \frac{3}{2}C(Opt).$$

$$C(S) \leq C(M) + C(T) \leq \frac{3}{2}C(Opt).$$

d'où :

$$C(S) \leq \frac{3}{2}C(Opt).$$

Exemple 4.2. Appliquant l'algorithme 2 sur cet exemple :
1- On a un deux arbres à deux dépôts :

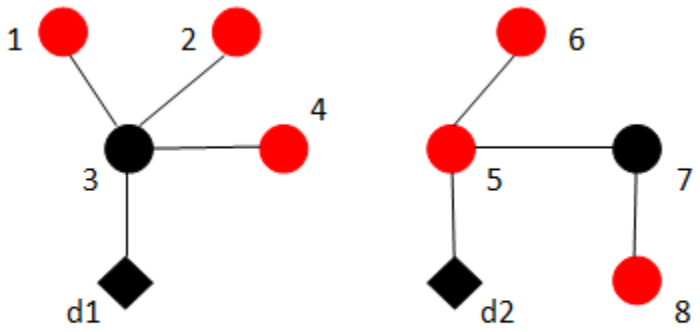


FIGURE 4.5

2- Trouvons un couplage partiel de coût minimum de cardinalité $\frac{|V_0|-2}{2}$, à l'aide de tous les sommets de degré impair
Dans notre cas on a 6 sommets de degré impair (1,2,4,5,6,8) d'où $\frac{|6|-2}{2} = 2$.

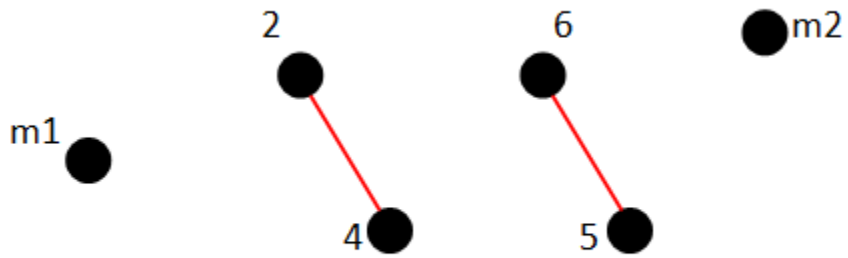


FIGURE 4.6

3- Ajoutons toutes les arêtes du couplage partiel à l'arbre pour obtenir un graphe eulérien et reliants les deux dépôts par une arête de coût 0 . On obtient la chaîne suivante :(m_1 3 4 2 3 d_1 d_2 5 6 5 7 m_2) .

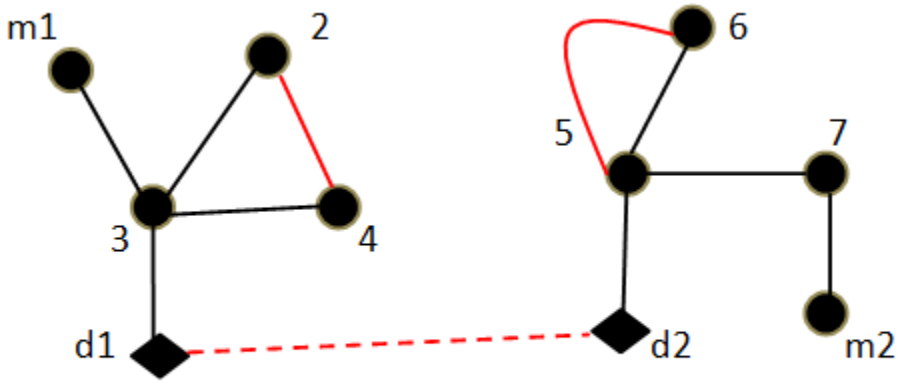


FIGURE 4.7

4- Raccourcir le tour eulérien pour obtenir un cycle hamiltonien, on appliquant le short-cutting comme suit :

{m1 3 2 4 ~~3~~ d1 d2 5 6 ~~5~~ 7 m2}

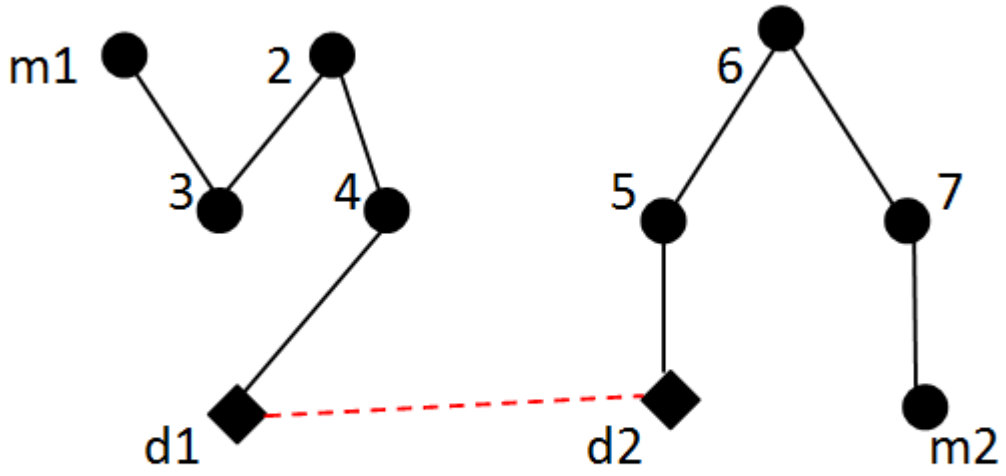


FIGURE 4.8

Ainsi on obtient la chaîne hamiltonnienne suivante : (m_1 3 2 4 d_1 d_2 5 6 7 m_2) puis, on supprime l'arête de coût zéro.

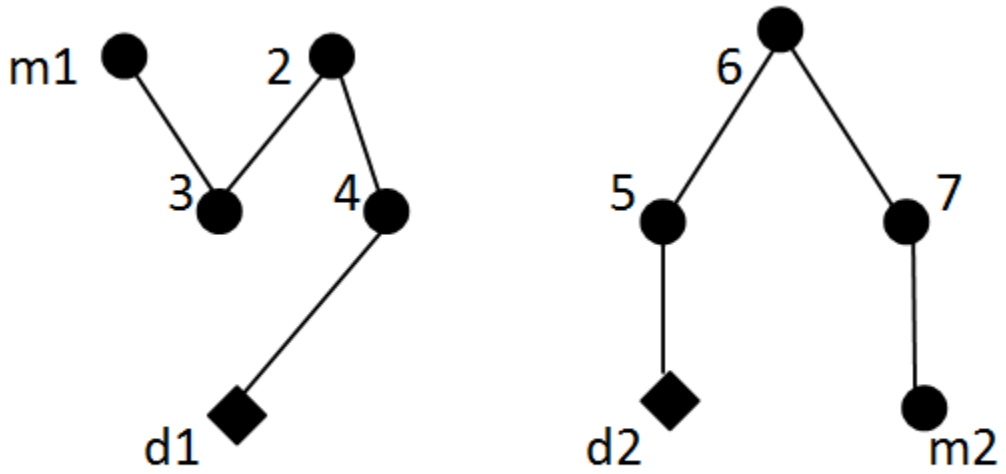


FIGURE 4.9

On obtient les deux chaînes hamiltoniennes suivantes :
(d_1 4 2 3 m_1) et (d_2 5 6 7 m_2)

4.5 Problème du sac à dos

4.5.1 Définition du problème :

Le problème du sac à dos (knapsack problem) est l'un des problèmes d'optimisation combinatoire les plus étudiés, il est défini par :
Étant donné un sac à dos de capacité K et une liste d'objets avec leurs poids et leurs valeurs. L'objectif est de trouver les objets à mettre dans le sac de manière à maximiser la valeur totale tout en respectant la contrainte de capacité du sac.

Modélisation :

Soient n objets numérotés de 1 à n possédant chacun un poids w_i et une valeur v_i et un sac de capacité K tel que pour chaque objet i, on a :

$$x_i = \begin{cases} 1 & \text{si l'objet i est pris} \\ 0 & \text{sinon} \end{cases}$$

et la somme des poids associés aux objets ne dépasse pas la capacité K du sac c'est à dire :

$$\sum_{i=1}^{n} w_i x_i \leq K$$

D'où le problème se modélise de la manière suivante :

$$Z(x) = \sum_{i=1}^{n} v_i x_i \longrightarrow max$$
$$\sum_{i=1}^{n} w_i x_i \leq K$$
$$x_i \in \{0, 1\}, i \in \{1, ..., n\}$$

Ce problème trouve des applications dans divers domaines tels que la logistique, la planification de la production, la finance et la gestion des stocks, entre autres .

Le problème est connu pour être NP-difficile [8] ,ce qui signifie qu'il n'existe pas d'algorithme polynomial pour le résoudre de manière exacte en un temps raisonnable.
Cependant, il existe des algorithmes d'approximation pour des solutions proche de l'optimum.

4.6 Approximation du problème du sac à dos

Entrées : Un ensemble de n couples de nombres entiers (w_i, v_i), Un entier K (capacité du sac).

4.6 Approximation du problème du sac à dos

Question : Remplir le sac en maximisant le bénéfice total c'est-à-dire, sélectionner des objets tels que :

$\sum_i w_i \leq K$

Algorithm 5 Algorithme du Sac à dos [12]

On introduit la densité de l'objet i comme $\theta_i = \frac{v_i}{w_i}$

- On trie les objets par ordre décroissant de la densité .
- Mettre les objets dans cet ordre l'un après l'autre tant qu'ils rentrent dans le sac. Ainsi, on construit de proche en proche une solution gloutonne S_1 jusqu'à ce qu'un dernier objet ne rentre plus dans le sac.
-Construire une seconde solution S_2 avec l'objet qui a le plus grand profit (valeur).
-Prendre la solution S qui réalise le maximum entre S_1 et S_2.

4.6 Approximation du problème du sac à dos

Théorème 4.3. Pour chaque instance I du problème, on a $S \geq \frac{1}{2}S^*$, où S^* est la solution optimale.

Preuve. On note par i_1 l'indice du 1^{er} objet non sélectionné dans S_1.
On note par D le sous ensemble des objets $S_1 \cup i_1$
On a la propriété de borne supérieure suivante :

$$S^* \leq \sum_{i \in S_1} v_i + v_{i_1}$$

On considère les objets de la solution $S_1 et S_2$
$v_{(S_1 \cup S_2)} \geq v_D \geq S^*$, car $S_2 \geq v_{i_1}$
On a : $S = max(S_1, S_2)$ est la solution obtenue par l'algorithme.

$$S = max(\sum_{i \in S_1} v_i, v_{S_2})$$

Comme 2 $max(X, Y) \geq X + Y$ donc :

$$2\ max(\sum_{i \in S_1} v_i, v_{S_2}) \geq (\sum_{i \in S_1} v_i, v_{S_2}) \geq S^*$$
$$2.S \geq S^*$$

$$S \geq \frac{1}{2}S^*$$

Exemple 4.3. Soit un sac de capacité $K = 10$ et un ensemble de 10 objets tel que :

Index	1	2	3	4	5	6	7	8	9	10
(w_i, v_i)	(2,6)	(2,7)	(6,10)	(3,10)	(8,17)	(5,12)	(4,8)	(8,20)	(9,24)	(4,9)

4.6 Approximation du problème du sac à dos

Le problème s'écrit :

$$\begin{cases} Z(X) = 6\mathrm{x}_1 + 7x_2 + 10x_3 + 10x_4 + 17x_5 + 12x_6 + 8x_7 + 20x_8 + 24x_9 + 9x_{10} \longrightarrow max \\ 2\mathrm{x}_1 + 2x_2 + 6x_3 + 3x_4 + 8x_5 + 5x_6 + 4x_7 + 8x_8 + 9x_9 + 4x_{10} \leq 10 \\ \mathrm{x}_i \in \{0,1\} \text{ , i=1,...,10} \end{cases}$$

Première étape :

On a la densité de l'objet i est $\theta_i = \frac{v_i}{w_i}$

On calcule la densité pour chaque objet :

Index	1	2	3	4	5	6	7	8	9	10
(w_i, v_i)	(2,6)	(2,7)	(6,10)	(3,10)	(8,17)	(5,12)	(4,8)	(8,20)	(9,24)	(4,9)
θ_i	3	3,5	1,66	3,33	2,12	2,4	2	2,5	2,66	2,25

Deuxième étape : On ordonne par ordre décroissant de densité tous les objets.

Index	2	4	1	9	8	6	10	5	7	3
θ_i	3,5	3,33	3	2,66	2,5	2,4	2,25	2,12	2	1,66

Troisième étape :

Objet 2 : On le met dans le sac vide, le poids du sac est alors de 2 est inférieur à 10 ;

Objet 4 : On le met dans le sac car 2+3 = 5 est inférieur à 10 ;

Objet 1 : On le met dans le sac car 2+3+2 = 7 est inférieur à 10 ;

Objet 9 : On ne le met pas dans le sac car le poids total (16) dépasserait 10.

La solution trouvée est donc $X = (1,1,0,1,0,0,0,0,0,0,0)$, $Z(X) = 6 + 7 + 10 = 23$

Donc S_1= 23

On calcule une autre solution avec l'objet du plus grand profit (valeur).

Ici on a l'objet 9 qui a pour profit 24 et un poids de 9, donc $S_2 = 24$.

On choisit comme solution le max(S_1, S_2), c'est -à dire :

S= max(S_1, S_2) =max$(23, 24) = 24$.

CONCLUSION GÉNÉRALE

Dans ce manuscrit, nous avons étudié l'approximation de quelques problèmes d'optimisation combinatoire tels que le problème du cycle hamiltonien, le problème de la chaîne hamiltonienne à deux dépôts et le problème du sac à dos.

Ces problèmes sont NP-difficiles, donc pour les grandes tailles, il n'existe pas d'algorithmes polynomiaux pour les résoudre.
Nous avons appliqué l'algorithme de Christofides pour le cycle hamiltonien puis adapté cet algorithme au cas de chaîne hamiltonienne à deux dépôts.

Pour cela, dans le cas de cycle hamiltonien on cherche un arbre de coût minimum et pour la chaîne hamiltonienne à deux dépôts, on cherche une forêt à deux arbres dans le graphe .
Cet algorithme admet un rapport d'approximation de $\frac{3}{2}$.

A la fin, nous avons présenté un algorithme d'approximation de rapport $\frac{1}{2}$ pour le sac à dos.
Tous ces rapports sont des rapports d'approximation standard.

Conclusion générale

L'approximation polynomiale s'avère être une approche prometteuse pour résoudre des problèmes d'optimisation combinatoire.
Elle permet d'obtenir des solutions proches de l'optimum avec une complexité temporelle raisonnable. De plus, des recherches supplémentaires peuvent être prises pour améliorer davantage ces approximations et développer de nouvelles approches pour résoudre ces problèmes de manière efficace et précise.

RÉSUMÉ

En résumé, ce mémoire a mis en évidence l'importance de trouver des approximations efficaces pour les problèmes d'optimisation complexes tels que le cycle hamiltonien et le sac à dos. Bien que ces approches ne garantissent pas toujours la solution optimale, elles offrent des résultats satisfaisants dans de nombreux cas pratiques.

BIBLIOGRAPHIE

[1] Alfred V Aho and John E Hopcroft. *The design and analysis of computer algorithms*. Pearson Education India, 1974.

[2] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. 1976.

[3] Marc Demange. Mathématiques et informatique.

[4] MC Enseignants-chercheurs permanents Wojciech Bienia, INPG Michel Burlet, MC UJF, Doctorants Pascal Chanas, Yann Kieffer, Post-Doctorants Sylvain Durand, Olivier Goldschmidt, Barry Guiduli, Rudi Pendavingh, Visiteurs Claude Benzaken, et al. Graphes et optimisation combinatoire.

[5] Bruno Escoffier. *Approximation polynomiale de problèmes d'optimisation : Aspects structurels et opérationnels*. PhD thesis, Ph. D. Thesis, Laboratoire LAMSADE, Paris Dauphine, 2005.

[6] Bruno Escoffier and Vangelis Th Paschos. Structures des classes d'approximation : un état de l'art. 2007.

[7] Bernhard Korte, Jens Vygen, Jean Fonlupt, and Alexandre Skoda. *Optimisation combinatoire*. 2010.

[8] Vangelis Th PASCHOS. *Optimisation combinatoire 1 : concepts fondamentaux*. Hermes science publications, 2005.

[9] Sivakumar Rathinam and Raja Sengupta. 32–approximation algorithm for two variants of a 2-depot hamiltonian path problem. *Operations Research Letters*, 38(1) :63–68, 2010.

[10] Cormen Thomas H, E Charles, Rivest Ronald L, Stein Clifford, et al. Introduction to algorithms third edition, 2009.

[11] Eric W Weisstein. Icosian game. *https ://mathworld. wolfram. com/*, 2003.

[12] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.