الجمـــــهوريـــــة الجـــزائريـــــة الديـــمقراطيـــــة الشعبيـــــة
RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
وزارة التعـلـيـــــم العـالـــــي و البـــحث العلـــــــــمي
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE
جـــــامعـة عمـــــــار ثليـجي بالأغـــواط
UNIVERSITÉ AMMAR TELIDJI LAGHOUAT

كـلـيـــــــة العلـــــــــوم
FACULTÉ DES SCIENCES
DÉPARTEMENT DE MATHÉMATIQUE ET INFORMATIQUE

MÉMOIRE DE MASTER

DOMAINE : MATHÉMATIQUES ET INFORMATIQUE **(MI)**
FILIÈRE : INFORMATIQUE
OPTION. : RÉSEAUX, SYSTÈMES ET APPLICATIONS RÉPARTIS **(ReSar)**

Présenté par :
CHAKHOUM MOHAMED LAMINE

Thème :

Coloriage de Graphe: Etude Expérimentale

Soutenu publiquement devant le jury composé de :

Mme.Chettih Atika	Maître-assistant (A)	U. Laghouat (Président)
M.Attia Nehar	Maître-assistant (A)	U. Laghouat (Examinateur)
M.Guellouma Younes	Maître-assistant (A)	U. Laghouat (Examinateur)
Mme.Cherroun Hadda	Maître de Conférences (A)	U. Laghouat (Rapporteur)

ANNÉE UNIVERSITAIRE 2011/2012

Le coloriage de graphe

Chakhoum Mohamed Lamine

4 juillet 2012

Remerciements

Je veux tout d'abord remercier l'Encadreur Mme Hadda CHERROUN, pour ses conseils et ses soutiens.

Je remercie les membres du Jury pour avoir accepté de participer à ce jury et d'avoir passer du temps à examiner mon travail.

Un grand merci pour tout les enseignants du département informatique pour leur cours et conseilles tout au long de ces cinq ans.

Je tiens à remercier mes collègues pour leur soutien et plus particulièrement Kerrache Chaker, Biaa Mohamed, Tedjani Jamel, et Allaoui Ismail.

Mes remerciements vont enfin à ma famille : mes parents qui se sont toujours intéressés à mes travaux et qui m'ont toujours poussé à persévérer, mes frères et toute la famille.
Et tous ceux qui ont participé de prés ou de loin à la réalisation de ce travail.

Résumé

Ce travail vise à analyser expérimentalement un algorithme exacte de coloriage de graphe, dans cet algorithme le modèle de coloriage de graphe a été utilisé pour résoudre le problème d'ordonnancement des tâches [11], le but est d'évaluer ses performances pour le coloriage de graphe propre.
L'algorithme se base sur un schéma de séparation et évaluation(Branch-And-Bound), Nous avons fait une implémentation efficace en langage C, et mesurer l'algorithme sur une ensemble d'instance de graphe choisit depuis un Benchmark [8]

Abstract

This work aims to analyse experimentaly an exact algorithm of graph coloring, this algorithm use a model of graph coloring to solve the problem of scheduling.
The objective is to evaluate its performance for the problem of graph coloring, the algorithm use a Branch-And-Bound method, we will make a new implementation in C language and evaluat its performance with some instance of graph choosed from a Benchmark [8].

Table des matières

Table des figures

Introduction

Le problème de coloration de graphes est un problème central de l'optimisation combinatoire. Il a de nombreuses applications pratiques, telles que les problèmes d'ordonnancement, d'emploi du temps, ou encore d'allocation de fréquences.De nombreuses méthodes approchées sont proposées pour résoudre le problème de coloration.
Il existe assez peu de méthodes exactes qui s'avèrent efficaces pour traiter ce problème : stratégies d'énumération implicite [16], génération de colonnes et programmation linéaire [10], branch-and-bound [4] et branch-and-cut [5].
En plus il y aussi la méthode metaheuristique, les métaheuristiques sont des algorithmes pouvant être appliqués à la résolution d'un grand nombre de problèmes d'optimisation. Elle ont reçu un intérêt grandissant et ont montré leur efficacité dans de vastes domaines d'application en résolvant de nombreux problèmes d'optimisation.
Pour le problème de coloriage nous avons la méthode Tabou et la méthode de Recherche Local.
Les métaheuristiques sont des algorithmes d'optimisation visant à résoudre des problèmes d'optimisation difficiles, autrement dit une métaheuristique est un processus itératif qui subordonne et guide une heuristique, en combinant intelligemment plusieurs concepts pour explorer et exploiter tout l'espace de recherche.

Dans notre travail, on s'interiesse à l'analyse d'un algorithme exacte utilisant une métaheuristique, Cet algorithme est conçu initialement pour donner une solution à un problème d'ordonnancement des tâches à travers un modèle de coloriage de graphe.
L'algorithme est basé sur un schéma à séparation et évaluation (branch-and-bound) [1], [11].

En effet, cet algorithme a donné de très bons résultats dans son contexte d'usage, le but de notre recherche est de mesurer son efficacité à travers une nouvelle implémentation, et faire un jeu de test sur des instances de taille plus importante et voir jusqu'à quelles limites cet algorithme donnent des résultats efficaces en temps raisonnable.
Nous comptons sur deux aspects :
Une implémentation efficace en langage C ; et une étude expérimentale sur ensemble d'instances tirés d'un benchmark connu [8].
Le présent document est organisé de la façon suivante :

1. Le chapitre 1, Notion fondamentales sur les graphes, vise à introduire des notions de base de la théorie des graphes au lecteur, ainsi qu'un rappel de domaines d'applications de modèle de coloriage de graphe.
2. Le chapitre 2, Les Algorithmes de coloriage, nous présentons une taxonomie des algorithmes de coloriage parmi les plus connus.
3. Dans le chapitre 3, L'approche branch-and-bound et ses performances, nous décrivons en détail l'algorithme Branch-and-bound, sujet de l'étude, et nous présentons les différents mesures réalisées et commentons les résultats obtenus.

Chapitre 1

Notion fondamentales sur les graphes

Dans ce chapitre, on rappelle les notions fondamentales de la théorie des graphes qui constituent un prérequis pour la compréhension du problème de coloriage des graphes.

1.1 Définition d'un graphe

Un graphe $G = (V; E)$ est défini par les deux ensembles V et E, telque V est l'ensemble des sommets (nœuds) [1]. E est l'ensemble des arêtes. Une arête e de l'ensemble E est définie par une paire non-ordonnée de sommets, appelés les extrémités de e. Si l'arête e relie les sommets v et u, on dira que ces sommets sont adjacents, ou incidents avec e, ou encore que l'arête e est incidente avec les sommets v et u. Un sommet v est adjacent à un sommet u s'ils sont reliés par une arête e. Une arête e est une boucle si ses deux extrémité sont confondues. Un degré de sommet v est le nombre d'arêtes incidentes avec ce sommet, on note avec $d(v)$. Un graphe dont tous les sommets ont le même degré est dit régulier. Si le degré commun est k, on dit alors que le graphe est k-régulier[2].

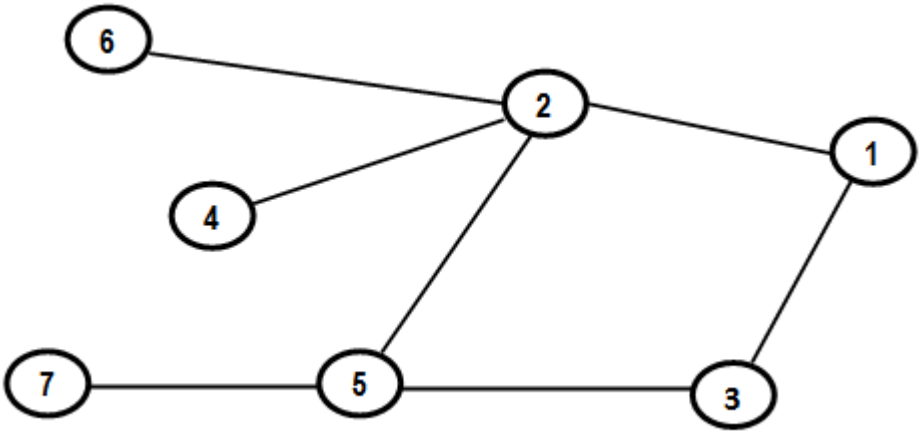

FIGURE 1.1 – Exemple d'un graphe G

1.2 Les différents concepts de graphes

Soit $G = (V; E)$ un graphe . $V = \{1, 2, 3, 4, 5, 6, 7\} E = \{(1, 2), (1, 3), (2, 5), (3, 5), (2, 4), (5, 7), (2, 6)\}$

Ordre du graphe

On appelle ordre d'un graphe le nombre de sommets (n) de ce graphe [14]. dans notre exemple $n = 7$

Graphe simple

Un graphe G est dit simple si pour tout paire de sommets v et u de V il y a au plus une arête incidente avec v et u.

Multi-graphes

Les graphes contenant plus d'une arête entre deux sommets ou des boucles sont appelés multi-graphes.

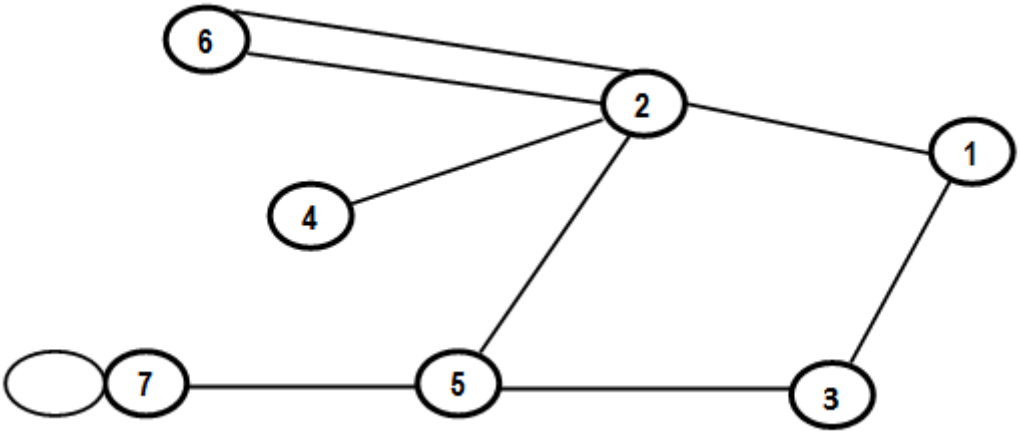

FIGURE 1.2 – Exemple d'un Multi graphe

Graphe partiel

$G' = (V'; E')$ est un graphe partiel de G si $V' = V$ et E' est un sous ensemble du E, Autrement dit, on obtient G' en enlevant une ou plusieurs arêtes au graphe G.

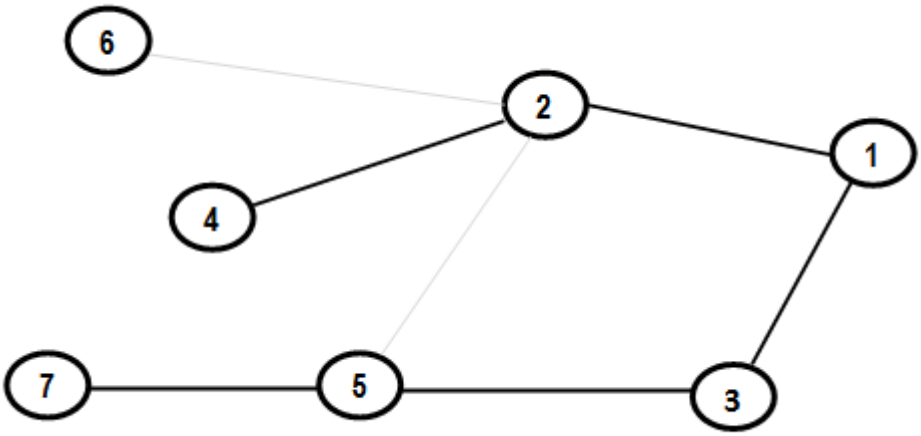

FIGURE 1.3 – Exemple d'un graphe partiel

Sous graphe

$G' = (V'; E')$ Est un sous graphe de G si V' est inclut dans V et E' est contient tous les arêtes qui ont les deux extrémités dans V'

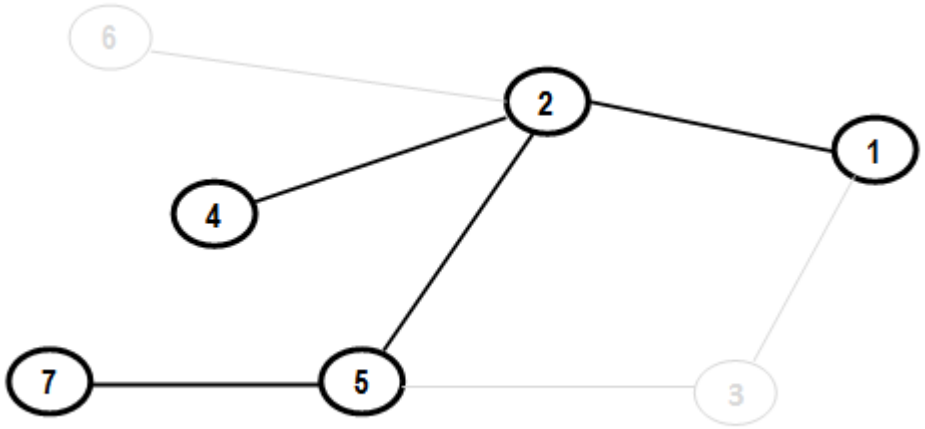

FIGURE 1.4 – Exemple d'un sous graphe

Graphe complet

Un graphe est dit complet si pour toutes paires de sommets, il existe une arête e qui les relie.

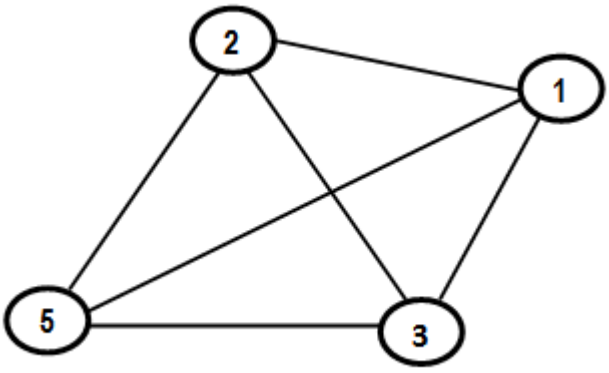

FIGURE 1.5 – Exemple d'un graphe complet

Clique

On appelle clique un sous-graphe complet de G, la clique maximale est la plus grande clique du G. noté par $\omega(G)$ [1]. autrement dit :

$$\forall \text{ v,u} \in \text{V}, \exists \text{ e} \in \text{E telque e=(u,v)}$$

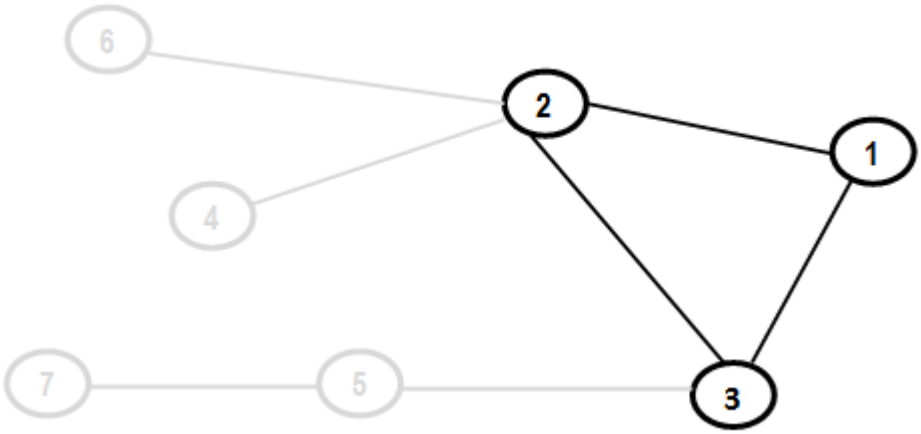

FIGURE 1.6 – Exemple d'une Clique

Stable

On appelle un stable un sous-graphe de G qui ne contient aucune arête, le stable maximal est le plus grand stable du G. noté $\alpha(G)$. autrement dit

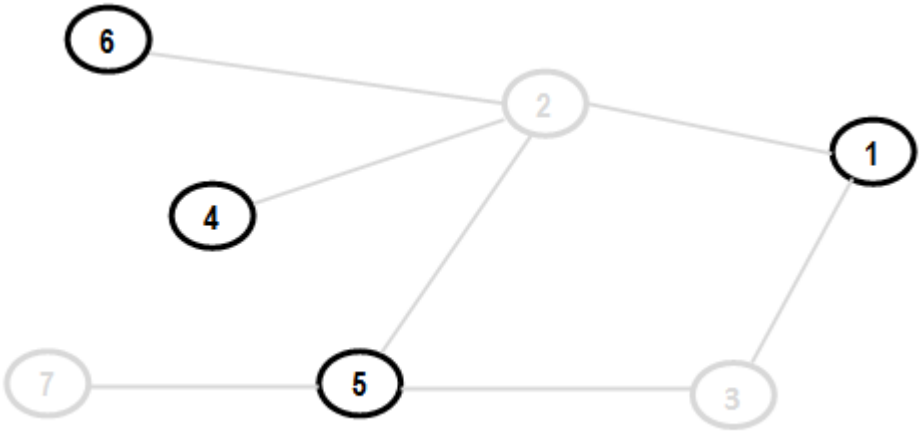

FIGURE 1.7 – Exemple d'un Stable

Chaine

Une chaine est une suite alternée des sommets et des arêtes sous la forme $C = v_0$ $e_1 v_1$ e_2 .... v_{k-1} e_k v_k, ou tels que pour $1 \leq i \leq k$, les deux extrémités de e_i sont v_{i-1} et v_i,on dit alors que la chaine est de longueur k.

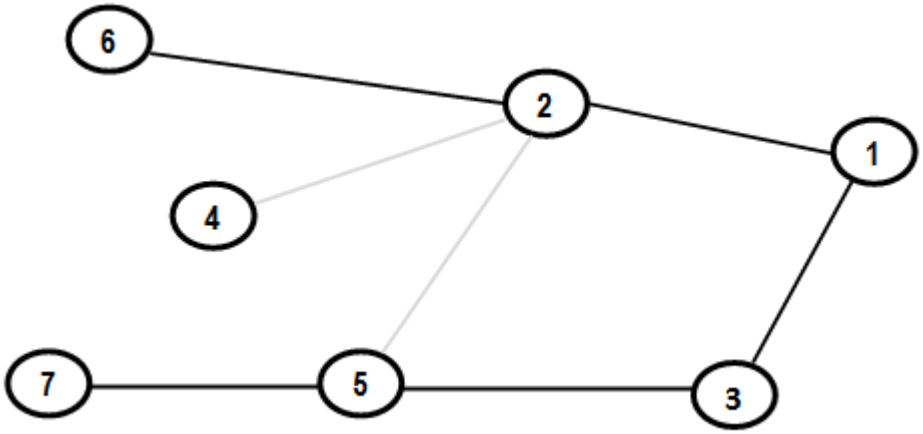

FIGURE 1.8 – Exemple d'une Chaine

Une chaîne est élémentaire si chaque sommet y apparaît au plus une fois.
Une chaîne est simple si chaque arête apparaît au plus une fois.
On appelle distance entre deux sommets la longueur de la plus petite chaîne les reliant.
On appelle diamètre d'un graphe la plus longue des distances entre deux sommets. [14]

Cycle

Un cycle est une chaine où les deux extrémités sont confondues.

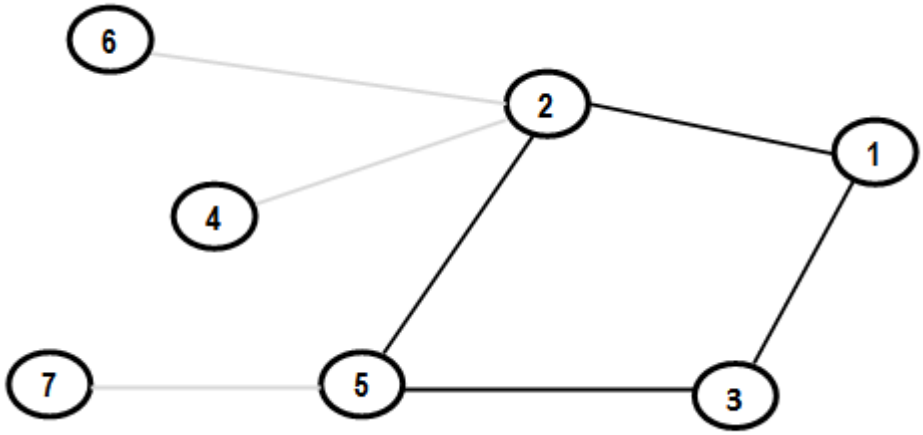

FIGURE 1.9 – Exmple d'un Cycle

Cycle eulérien

On dit qu'un cycle est eulérien si ce cycle passe une et une seule fois par toutes les arêtes.
Un graphe qui possède un cycle eulérien est dit graphe eulérien.
Un graphe est dit semi-eulérien s'il est possible de trouver une chaîne passant une et une seule fois par toutes les arêtes. [1]

Cycle hamiltonien

On dit qu'un cycle est hamiltonien si ce cycle passe une et une seule fois par tous les sommets.

Un graphe qui possède un cycle hamiltonien est dit graphe hamiltonien.

Un graphe est dit semi-hamiltonien s'il est possible de trouver une chaîne passant une et une seule fois par tous les sommets.

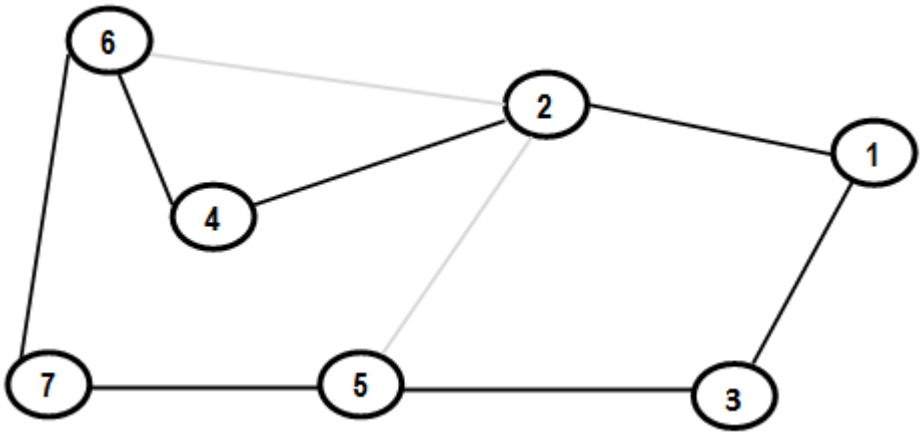

FIGURE 1.10 – Exemple d'un Cycle hamiltonien

Nœud isolé

Un nœud est dit isolé s'il n y a aucune arête qui est incident avec . [14]

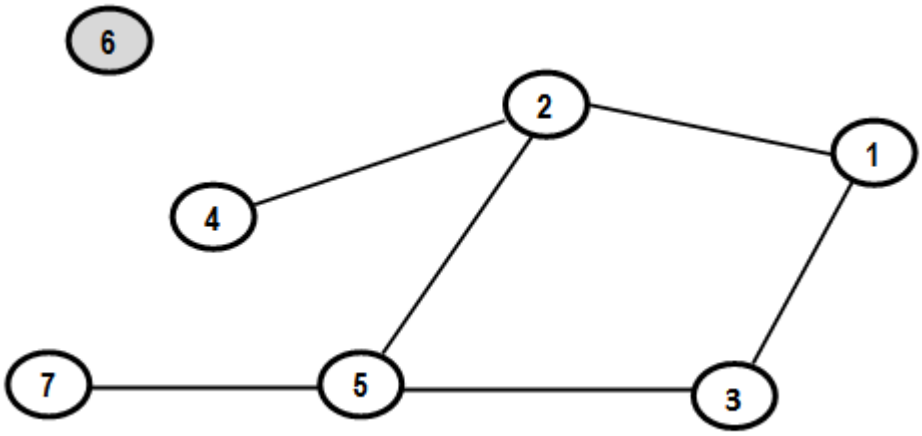

FIGURE 1.11 – Exemple d'un nœud isolé

La connexité d'un graphe

Un graphe est dit connexe si pour chaque paire de sommets, il existe une chaine qui relie ces deux sommets.

Graphe planaire

on appelle un graphe planaire un graphe où on peut le représenter dans un plan sans qu'il y ait intersection d'arêtes.

1.3 Arbres et arborescences

Arbre

Un graphe est dit arbre s'il est connexe et sans cycle. [14]

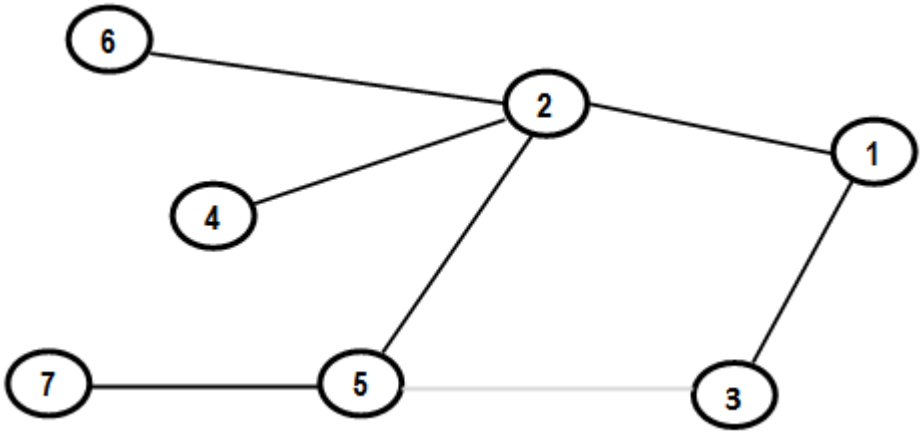

FIGURE 1.12 – Exemple d'une arbre

Foret

Une foret est un graphe sans cycles et non connexe. [14]

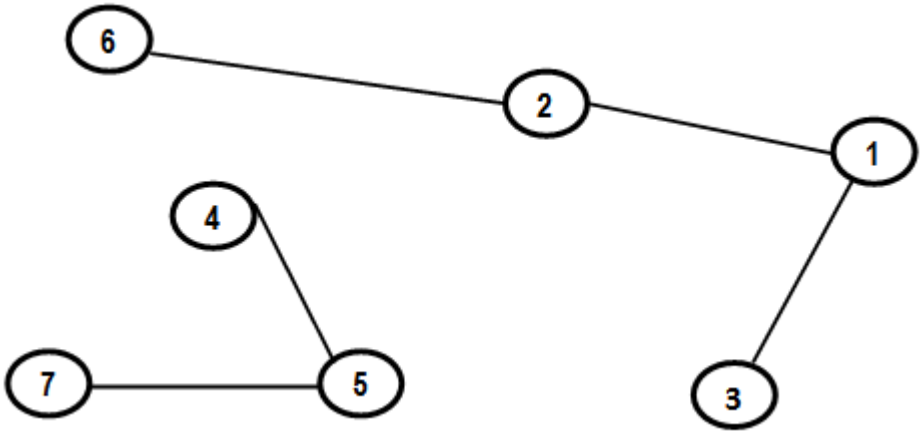

FIGURE 1.13 – Exemple d'une foret

Arbre couvrant

Appelé aussi arbre maximal,un est un graphe partiel qui est aussi un arbre.

1.4 Le coloriage des graphes

Le coloriage des graphes est un problème classique qui peut être adapté à une large variété d'applications.
le coloriage des graphes peut donc résoudre plusieurs problème dans le pratique et la théorique.
Soit G = (V ;E) un graphe.

1.4.1 Coloration des sommets

On appelle f une coloration propre du G, une fonction de l'ensemble V vers l'ensemble C de couleur, de telle sorte que deux sommets adjacents ne portent pas la même couleur.

$$f : V \longrightarrow C$$
$$\text{Si } \exists \text{ (v,u)} \in \text{E alors } f(\text{u}) \neq f(\text{v})$$

Un coloriage de sommets avec k couleurs est donc une partition de l'ensemble des sommets en k stables.

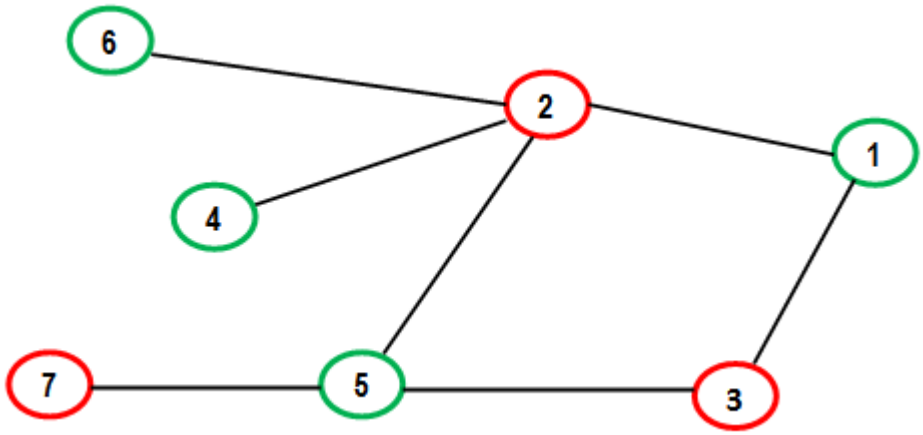

FIGURE 1.14 – Coloration des sommets

1.4.2 Coloration des arêtes

On appelle f une coloration propre du G, une fonction de l'ensemble E vers l'ensemble C de couleur, de telle sorte que deux sommets adjacents ne portent pas la même couleur. [14]

$f : E \longrightarrow C$

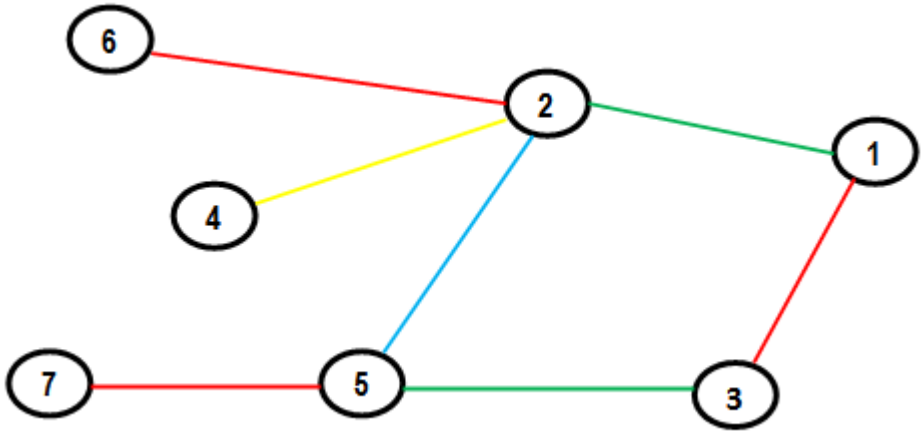

FIGURE 1.15 – Coloration des arêtes

1.4.3 Le nombre chromatique $\chi(G)$

On appelle nombre chromatique le nombre minimum de classes de couleur qui assure une coloration propre du G. [14]

1.4.4 L'indice chromatique χ'(G)

L'indice chromatique est le plus petit nombre de classes de couleur qui assure une coloration des arêtes du G.

1.5 Encadrement du nombre chromatique

On note

$G = (V, E)$ un graphe.

$\chi(G)$ le nombre chromatique.

$\omega(G)$ l'ordre de clique maximale.

$\alpha(G)$ le cardinal de stable de graphe.

$\Delta(G)$ le degré maximal des sommets de G.

N le nombre de nœud.

On cherche à trouvé des borne supérieurs ou inférieurs au nombre chromatique pour facilite le calcul de ce nombre,

1.5.1 MINORATION

Si H est un sous graphe de G alors

$$\chi(H) \leq \chi(G)$$

La clique maximale du graphe est un sous-graphe complet. Pour un graphe complet, $\chi(G)$ = le nombre de nœud. C'est bien claire que le nombre chromatique est supérieur ou égale au l'ordre de clique maximale.

$$\omega(G) \leq \chi(G)$$

On a aussi cette inégalité

$$\frac{n}{\alpha(G)} \leq \chi(G)$$

α(G) est l'ordre du clique maximal

1.5.2 MAJORATION

Il existe plusieurs méthodes pour majoré le nombre chromatique. La première idée consiste à utilisé N couleur, c-à-dire chaque sommet prend une couleur identique.
la méthode la plus utilisé est d'utiliser un algorithme heuristique pour colorier le graphe, le nombre obtenu présente une majoration du nombre chromatique.

Autre majoration c'est le l'ordre maximal des sommets du graphe, car le plus difficile sommet a coloré est le sommet d'ordre maximal qui nécessite Δ(G) + 1 couleur. Donc chaque voisin prend une couleur en plus sa couleur.

$$\chi(G) \leq \Delta(\mathrm{G})+1$$

Une autre borne difficile à calculer utilise le cardinale du plus grand stable.

$$\chi(G) \leq \mathrm{n} - \alpha(\mathrm{G})+1$$

1.6 Exemples d'applications

de nombreux problèmes théorique ou pratique, sont modélisées en utilisant le coloriage de graphe, où les sommets représentent les entités ou bien les éléments et les arêtes représentent les contrainte et les liens entre ces entités, On cherche toujours à minimiser le nombre de couleurs utilisées, car en général il représente le cout (en temps par exemple) [1].
Dans ce qui suit on donnera quelques exemples d'applications théorique et pratiques :

1.6.1 Ordonnancement

Le problème d'ordonnancement est parmi les problèmes qu'on peut le modéliser par le coloriage de graphe, que ce soit l'ordonnancement des vols d'avions ou l'ordonnancement des taches bi-processeurs. Soit n taches à exécuter, Ces taches vont représentées les nœuds de notre graphe, une arête entre deux nœuds implique que les deux taches ne peuvent pas être exécuter au même temps, si chaque tache prend une unité de temps pour exécuter, le nombre chromatique représente alors le nombre minimal d'unité pour exécuter l'ensemble des taches, les différents colorations de graphe vont nous montrer les variantes possibles pour exécuter ces taches où chaque classe de couleurs est une ensemble de tache qui vont exécuter en parallèle.

1.6.2 Allocation des registres

Dans un compilateur, l'allocation de registres est une étape importante. Elle vise à choisir dans quel registre du processeur sera rangée chaque variable lors de l'exécution du programme compilé. Les opérations sur des valeurs rangées dans des registres sont considérablement plus rapides que celles sur des valeurs en mémoire vive. Mais les registres sont en nombre très limité, et souvent trop faibles pour ranger toutes les variables d'un programme. Il est donc crucial de choisir avec pertinence les variables qui doivent résider dans les registres à chaque étape de l'exécution. Alors on commence par faire comme si l'on disposait d'un nombre illimité de registres, puis l'on attribue à ces registres virtuels des registres réels ou des emplacements en mémoire. Une méthode classique consiste à ramener le problème au coloriage du graphe d'interférence des variables. Les sommets de ce graphe sont les variables du programme, et deux sommets sont reliés par une arête si les variables correspondantes interfèrent.

On dit que deux variables interfèrent lorsque l'une est définie pendant que l'autre est active (c'est-à-dire susceptible d'être utilisée dans la suite de l'exécution). Deux variables qui interfèrent ne peuvent pas être placées dans le même registre. Attribuer k registres aux variables équivaut à colorier avec k couleurs le graphe d'interférences. Les contraintes sur registres, spécifiques à la machine cible peuvent être représentées directement sur le graphe d'interférences.

Une variante du problème considère que l'on a seulement R registres physiques utilisables, et le problème devient une coloration du graphe d'interférence avec R couleurs. L'astuce est de stocker temporairement la valeur de registres virtuels peu utilisés en mémoire vive. Une fois mis en mémoire, on peut les enlever du graphe d'interférence : si on arrive à profiter bien de la nouvelle situation, le graphe devient R-coloriable. Il va donc falloir choisir des registres virtuels à mettre dans la mémoire (on utilise le terme 'spiller') : le problème étant -complet, il n'existe pas d'algorithmes efficaces pour choisir ces registres à spiller.

1.6.3 Assignation des fréquences

Dans un réseau cellulaire, l'accès des utilisateurs au réseau se fait par voie hertzienne via un transmetteur. Chaque transmetteur couvre une certaine zone, la communication est établie sur un canal de transmission correspond à une gestion partagée des fréquences dont le but est de pouvoir augmenter le nombre d'utilisateurs connectés en même temps à un même transmetteur, deux communications distinctes (avec le même transmetteur ou avec deux transmetteurs différents) ne peuvent pas avoir lieu sur le même canal si les utilisateurs sont trop proches l'un de l'autre, car leurs communications se brouilleraient. Le problème de l'allocation de fréquences consiste alors à attribuer à chaque transmetteur un nombre suffisant de fréquences,
Alors nous pouvions représenter le réseau cellulaire par un graphe dont les sommets sont les transmetteurs, et une arête signifie qu'il existe une possibilité d'interférence entre ces deux transmetteurs. [15]

1.6.4 Emploi du temps

La réalisation d'un emploi du temps consiste à ranger ou affecter les cours ou/et les examens dans des jours et des horaires bien définis , cependant il existe des contraintes,telles que :

- Deux cours, ou plus, assuré(s) par un même enseignant ne peuvent pas avoir lieu dans la même période.

- Deux cours, ou plus, qui requiert la même salle, ne peuvent pas avoir lieu dans la même période.

- Deux cours, ou plus, planifiés dans la même période ne peuvent pas avoir lieu dans la même salle.

Le coloriage de graphe résoudre le problème, où les cours (les examens) sont les nœuds, une arête entre deux nœuds implique l'existence d'au moins une contrainte entre les

deux cours,
Deux nœuds ont des couleurs différentes ne peuvent pas avoir lieu dans la même période, par contre, tout les nœuds qui ont la même couleur peuvent avoir lieu dans la même période.

1.6.5 Test des circuits imprimés

Ce problème est de tester les circuits imprimés sur une carte, pour les courts-circuits (causés par des lignes de soudure) , dans ce cas on représente les files par des nœuds et les arêtes vont représentent l'existence potentiel de risque pour un court-circuit entre les files.
L'objectif donc est de diviser la file en des ensembles où chaque ensemble a une classe de couleur, afin de tester les files d'un tel ensemble au même temps, sans avoir des risque ce qui accélère le processus de test.

1.6.6 Compression des images

Les images bitmap sont partitionnées en régions connectées, on cherche à distinguer, pour chaque pixel, la région à laquelle il appartient.
Chaque pixel a un code mot, chaque région est composé de l'ensemble des pixels qui ont le même code mot, alors on peut formulé par un problème de coloriage.

1.6.7 Problème du pécheur

Un autre problème connu est le problème de pécheur qui possède un ensemble de type de poisson, mais ces poisson ne sont pas tous compatible à cause de la loi proie-prédateur, donc pour connaitre le nombre minimal des compartiments, on modélise avec le coloriage, on obtient :
Les nœuds sont les différents types de poisson, et deux extrémité d'une arête représentent de types incompatible, Le nombre de couleurs est donc le nombre de compartiments nécessaire.

1.6.8 Problème du Carrefour

Le problème de carrefour est parmi les problèmes les plus connu, consiste à régler les feux de telle manière qu'on puisse répartir les flux de circulation en groupes de voitures qui ne vont pas se croiser. Ainsi, les flux d'un même groupe auront le feu vert simultanément, alors que, pendant cette période, les autres flux auront le feu rouge.on cherche à régler les feux pour que le temps d'attente moyen pour chaque voiture soit minimal.

le problème est modéliser par un graphe où :

chaque flux de circulation possible représente un sommet.

une arêtes indique que les flux de ce deux sommets se croisent.

la coloration de graphe résoudre le problème, et le nombre chromatique représente le temps d'attente minimal.

Chapitre 2

Les Algorithmes de coloriage

Le problème de coloriage est parmi les problèmes qui appartiens à la famille $\mathcal{NP}$-complets. Plusieurs algorithmes de coloriage ont été proposés, certains donnent une solution exacte et autre donnent une solution approchée, le but est de calculer le nombre chromatique, pour les algorithmes exact le temps nécessaire pour effectuer le calcul sera énorme avec des graphes de grand taille, par contre les solutions approché peut être utilisé dans un temps raisonnable.
Dans ce chapitre, nous allons décrire les familles d'algorithmes de coloriage selon leur complexité, dans la section suivante nous rappellerons les quatre grandes classes de problèmes.

2.1 Classes de complexité

nous présenterons quatre classes de complexité dont on parle le plus souvent : classe P, classe $\mathcal{NP}$, classer $\mathcal{NP}$-complet et $\mathcal{NP}$-difficile.

2.1.1 Classe P

P est la classe de tous les problèmes de décision qui peuvent être résolus par un algorithme polynomial.

2.1.2 P-complet

Un problème décisionnel appartient à ce classement s'il fait partie de la classe P et si tout problème P dans P peut s'y réduire en temps poly-logarithmique en utilisant un ordinateur avec un nombre polynomial de processeurs. En pratique, si X est un problème dans P alors il appartient à P et pour tout problème Y dans P, il existe les constantes c et k telles que Y puisse être réduit en X en $\theta((log(n)^c)$ avec un ordinateur qui possède $\theta(n^k)$ processeurs en parallèle.

2.1.3 Classe $\mathcal{NP}$

$\mathcal{NP}$ (Non-déterministic Polynomial time) est la classe de tous les problèmes de décision dont la solution peut être vérifiée en temps polynomial, i.e. si l'on nous donne une solution certifiée, il est possible de vérifier que cette solution est correcte en un temps polynomial par rapport à la taille de l'entrée.

2.1.4 Classe $\mathcal{NP}$-complet

Un problème $\mathcal{NP}$-Complet est un problème dans $\mathcal{NP}$ au moins aussi difficile que tout autre problème de $\mathcal{NP}$.

- Il existe une réduction polynomiale qui permet de décrire n'importe quelle instance de n'importe quel problème de $\mathcal{NP}$ comme une instance de $\mathcal{NP}$-Complet.
- Si on sait résoudre toutes les instances d'un problème $\mathcal{NP}$-complet on sait résoudre toutes les instances de tous les problèmes $\mathcal{NP}$.

2.2 Méthodes heuristique pour le coloriage

On appelle une méthode heuristique, un algorithme qui nous donne une solution réalisable pour un problème donnée dans un temps réduit, Il ne garantit pas que la solution soit optimale. Ces méthodes sont utilisées pour éviter le recours à des algorithme exactes où la complexité est exponentielle. Les algorithmes exacts peuvent actuellement résoudre uniquement des graphes aléatoires de petite taille, avec jusqu'à 80 sommets

[12].par contre les méthodes heuristiques peut être appliquées sur des graphes ayant plus de centaines de sommets [6].
Les heuristiques sont pertinentes pour calculer une solution approchée d'un problème et ainsi accélérer le processus de résolution exacte.
Les algorithmes heuristiques appartiennent essentiellement à trois approches de résolution :

1. Construction séquentielle ;
2. Recherches locales ;
3. Méthodes à base de populations ou distribuées ;

2.2.1 DSATUR

L'algorithme de DSATUR est un algorithme heuristique séquentiel de coloration de graphes créé par Daniel Brélaz en 1979, qui donne une solution approché dans un temps polynomial [3].

Degré de saturation : DSAT ( )

C'est une valeur associée à chaque nœud où pour un nœud X, le degré de saturation est :

- Si aucun voisin de X est coloré alors DSAT(X) = degré (X)
- Sinon DSAT(X) = le nombre de couleur utilisé dans les voisin de X.

Algorithme

1. Ordonner les sommets par ordre de degrés décroissant.
2. Colorer le sommet de degré maximum avec la couleur 1.
3. Choisir un sommet X non coloré ayant un DSAT maximum Si il y en a plusieurs, on choisit celui de degré maximum.
4. Colorier X avec la petite couleur possible.
5. Si tous les sommets sont coloriés alors Stop. Sinon aller à 3.

Complexité

L'algorithme commence par trier les sommets par rapport a leur degré, un algorithme de tri rapide a une complexité de $\theta\ (n \log n)$, et après on doit calculer le degré de saturation pour chaque nœud ce qui peut prendre n opérations au plus, donc une complexité $\theta(n)$, enfin colorer le sommet, au pire de cas a une complexité de $\theta(n-1)$, on a donc une somme égale à $\theta(n^2)$.

2.2.2 Welsh-Powell

L'algorithme de Welsh-Powell est le premiers algorithme pour la coloration de graphe, Il a été développés dans les années soixante, c'est un algorithme heuristique qui donne une solution approché dans un temps polynomial [13].

Algorithme

1. Ranger les sommets par ordre de degrés décroissants.
2. Attribuer au premier sommet X de la liste une couleur.
3. Suivre la liste en attribuant la même couleur au premier sommet Y qui ne soit pas adjacent à X.
4. Suivre la liste jusqu'à ce que la liste soit finie, en attribuant la même couleur à tout sommet non adjacent à un sommet déjà coloré par cette couleur.
5. Prendre une deuxième couleur pour le premier sommet non encore colorié de la liste.
6. Répéter les opérations 3 à 4.
7. Continuer jusqu'à avoir coloré tous les sommets.

Complexité

La première étape est le trie des sommets par rapport a leur degré, avec un algorithme de tri rapide sa prend une complexité de $\theta\ (n \log n)$,et puis on fait un parcours de la liste

au plus n fois, c'est donc une complexité de $\theta(n^2)$, alors l'algorithme a une complexité égale a $\theta(n^2)$.

2.2.3 Algorithme glouton

Le principe de l'algorithme glouton (greedy algorithm) : faire toujours un choix localement optimal dans l'espoir que ce choix mènera à une solution globalement optimale.[14]

Algorithme

- Input : Un graphe G et des couleurs 1,2,3,4. . .Les sommets de G sont nuémerotés de 1 à n ($s_1, s_2, ..., s_n$).
- Output : Une coloration valide du graphe G.
- Traitement : Pour i allant de 1 à n, affecter au sommet s_i la plus petite couleur non déjà affectée à ses voisins déjà coloriés (c'est-à-dire la plus petite couleur non déjà affectée à ceux des sommets $s_1, s_2, ..., s_{i-1}$ qui lui sont adjacents).

Remarque

l'ordre de numérotation a un grand influence sur la solution obtenue.

Exemple

Soit les deux graphes suivants

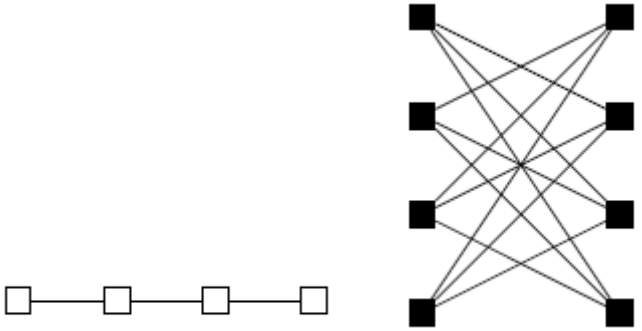

FIGURE 2.1 – Exemple d'application de l'algorithme gloton

Première numérotation :

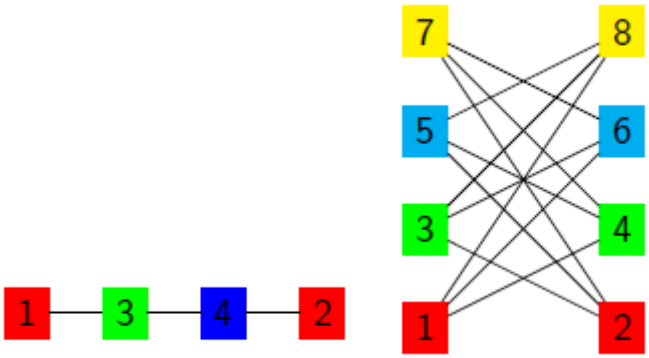

FIGURE 2.2 – coloration avec la première numérotation

Seconde numérotation :

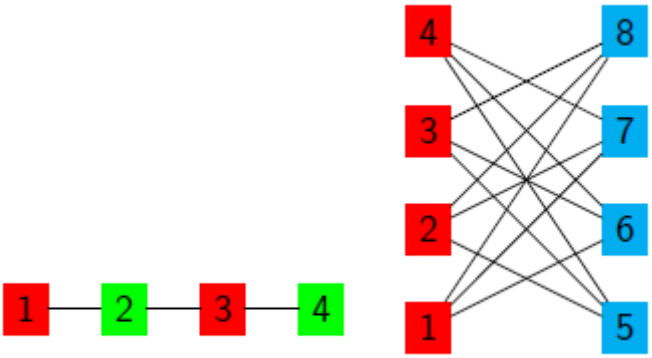

FIGURE 2.3 – coloration avec la deuxième numérotation

2.3 Méthode métaheuristique

Les métaheuristiques forment un ensemble de méthodes utilisées en recherche opérationnelle pour résoudre des problèmes d'optimisation réputés difficiles. Résoudre un problème d'optimisation combinatoire, c'est de trouver l'optimum d'une fonction, parmi un nombre fini de choix, souvent très grand [12].
les métaheuristiques permettent, dans des temps de calcul raisonnables, de trouver des solutions, peut-être pas toujours optimales, en tout cas très proches de l'optimum.
Les métaheuristiques sont généralement des stratégies de haut niveau qui guident une heuristique sous-jacente, plus spécifique, afin d'augmenter ses performances [12].

2.3.1 La méthode Tabou

La méthode Tabou est une technique de recherche dont les principes ont été proposés pour la première fois par Fred Glover dans les années 80, et elle est devenue très classique en optimisation combinatoire. Elle se distingue des méthodes de recherche locale simples par le recours à un historique des solutions visitées, de façon à rendre la recherche un peu moins " aveugle ". Il devient donc possible de s'extraire d'un minimum local, mais, pour éviter d'y retomber périodiquement, certaines solutions sont bannies, elles sont rendues " taboues " [2].

Le voisinage

L'espace de recherche Ω d'un problème de k-coloration $(G_{V;E}; k)$ comprend toutes les colorations possibles de G ; comme les configurations sont codées sous forme de vecteurs de couleurs, on obtient $|\Omega| = |V|^k$. Une fonction simple de voisinage $N : \omega \to 2^{\Omega} - \{\varnothing\}$ peut être définie comme suit :
étant donnée une k-coloration C, une k-coloration voisine C' peut être obtenue en changeant simplement la couleur c(i) d'un sommet en conflit i avec une nouvelle couleur c'(i). La transition de C à C' représente un mouvement (un pas) dans un processus de recherche, formellement noté par le couple i,c'(i) .
Ce voisinage se concentre sur les |CV| sommets en conflit, afin d'aider le processus de recherche à se focaliser sur des mouvements influents, en évitant les mouvements moins pertinents. Habituellement, il n'y a pas besoin de changer la couleur d'un sommet non conflictuel. La taille de notre voisinage est |(NC)| = (k-1) |CV| une taille considérablement plus petite que la taille d'un voisinage dans lequel tout sommet pourrait changer sa couleur (i.e. (k - 1) |V|). [12]

Evaluation rapide du voisinage complet

Pour choisir rapidement la meilleure coloration dans N(C), l'algorithme utilise un tableau γ avec |V| linges et k colonnes : un élément $\gamma_{i,j} = \gamma_{i,c'(i)}$ indique le nombre de conflits que le sommet i aurait si la couleur j = c'(i) lui avait été assignée. La différence du nombre de conflits qui serait induite par un mouvement (i,c'(i)) est $\gamma_{i,c'} - \gamma_{i,c(i)}$.
Comme ce voisinage considère seulement des sommets en conflit, le meilleur mouvement est recherché en passant par tous les éléments $\gamma_{i,j} avec i \in CV (i.e. (k-1) \mid VC \mid éléments de \gamma)$; on peut dire que la couleur c'(i) qui sera affectée à i vérifie c'(i)$\in argmin_j(\gamma_{i,j} - \gamma_{i,c(i)})$. Aprés avoir effectué un mouvement (i ; c'(i)) , γ peut être mis à jour en θ (|V|) opérations : il faut modifier uniquement les colonnes c(i) et c'(i).

Gestion de la liste Tabou

La liste Tabou est généralement vue comme une structure "first-in-first-out" (une file) qui contient des configurations ou des mouvements récents. Dans notre contexte, il est plus pratique de l'implémenter comme un tableau T de dimension $\mid V \mid \times k$ ou chaque élément correspond à un mouvement possible. Chaque fois qu'un mouvement (i,c'(i)) est exécuté, le sommet i reçoit une nouvelle couleur c'(i) et l'ancienne couleur de i i.e. c(i) devient interdite (Tabou) pour les prochaines T_l itérations (la durée Tabou est T_l).

Dans la pratique, chaque élément de T indique le nombre courant d'itération plus la durée Tabou T_l. Par conséquent, afin de vérifier si un mouvement (i,c'(i)) est Tabou ou non, $T_{i,c'(i)}$ est comparé avec l'itération courante.

Algorithme

Algorithm 1 La recherche Tabou pour k-Coloration

Entrée $grapheG, k \in IN^*$;

Valeur de retour f(C^*) ; le meilleur nombre de conflits jamais trouvé ;

Variables :C et C^* : la coloration courante et la meilleure coloration trouvée ; T et T_l : tableau Tabou et durée Tabou ; γ : tableau incrémental | V | × k

Debut

– I = 0

T = 0

C = une k-coloration aléatoire

$C^* = C$

initialiserγ

tant que (f(C) ≤ 1 et condition d'arrêt non atteinte)

– Sélectionner le meilleur mouvement $non - Tabu^a$ dans γ (Si plusieurs mouvements mènent au même meilleur nombre de conflits, un choix aléatoire est fait utilisant la fonction d'évaluation)

– $T_{I,c(i)}$ = I + T_l

– c(i) = c'(i) (effectuez le mouvement)

– f(C) = f(C) + $\gamma_{i,c'(i)} - \gamma_{i,c(i)}$

– Actualiser γ

– **si** (f(C) ≤ f(C*)) alors C^* = C

renvoyer f(C^*)

fin

2.3.2 Recherche locale

Les techniques de recherche locale associent un voisinage N(s) à chaque solution s ∈ S, espace des solutions, et construisent une séquence $s_0, s_1, s_2, ...$ de solutions où s_0 est une solution initiale généralement produite par un algorithme constructif simple, et où chaque s_i fait partie du voisinage N(s_{i-1}) de s_{i-1}. Toute solution dans le voisinage

N(s) de s est dite voisine à s. Divers critères d'arrêt peuvent être considérés, tel qu'un temps limite imposé ou un écart par rapport à une borne inférieure jugé suffisant. Une description générale des techniques de recherche locale est donnée ci-dessous.

Algorithme

Générer une solution initiale s et poser s* = s

Tant que aucun critère d'arrêt n'est satisfait

1. Générer une solution s' dans le voisinage N(s) de s.
2. Poser s = s' et mettre à jour s* si f(s) $\leq$ f(s*).
   - s* contient la solution optimale obtenue et f(s) représente la fonction économique de la solution s.

fait

2.4 les Méthodes exactes de coloriages

On appelle une méthode exacte toute méthode qui cherche la solution optimal d'un problème donnée.

2.4.1 Algorithme de Randall-Brown

Est un algorithme qui cherche une solution optimale Pour comprendre le principe de cet algorithme, on va définir Une solution partielle de niveau p, (p < n) est un coloriage du graphe G dans lequel seuls les sommets $v_1; ...; v_p$ sont coloriés.

Principe de l'algorithme

Soit K_{pq} une solution partielle de niveau p, avec q représente le nombre de couleurs utilisées, et p le nombre de sommets coloré.
A partir de K_{pq} on obtiendra toutes les solutions en procédant comme suit :

1. Choisir un nouvel sommet v_{p+1}.

2. Affecter à v_{p+1} les couleurs 1,2,...,q+1.
3. Eliminer les colorations non faisables.
4. Procéder de la même manière en utilisant les solutions partielles obtenues,

On prend les solutions qui utilise moins de couleur.

Complexité

L'opération essentielle dans cet algorithme est l'affectation de couleur au sommet, dans le pire des cas on trouve la complexité comme suit :

- Au début on a une coloration de p sommets avec q couleurs, donc au pire des cas q = p, on a alors $\prod_1^p k$ opérations caractéristiques.
- A l'étape suivante avec le nœud v_{p+1} on aura de 1,...,P+1 couleurs possibles, alors (p ! * p+1) c-à-d le nombre d'opérations sera égale à le nombre d'opération précédente multiplié par p+1 alors $\prod_1^{p+1} K$
- En général avec un graphe de n sommets on a une complexité de $\theta(n!)$

2.4.2 Méthode à séparation-évaluation (Branch-and-Bound)

La technique du Branch-and-Bound est une méthode algorithmique classique pour résoudre un problème d'optimisation combinatoire. Il s'agit de rechercher une solution optimale dans un ensemble combinatoire de solutions possibles. La méthode repose d'abord sur la séparation (branch) de l'ensemble des solutions en sous-ensembles plus petits. L'exploration de ces solutions utilise ensuite une évaluation optimiste pour majorer (bound) les sous-ensembles, ce qui permet de ne plus considérer que ceux susceptibles de contenir une solution potentiellement meilleure que la solution courante.
Appliquées à des problèmes $\mathcal{NP}$-difficiles, ces méthodes restent bien sûr exponentielles, mais leur complexité en pratique, lorsqu'elles sont bien paramétrées, est bien meilleure que pour une énumération complète. Elles peuvent donc pallier le manque d'algorithmes polynomiaux.
Dans les méthodes à séparation et évaluation,la séparation (branch) consiste à séparer un ensemble de solutions en sous-ensembles tandis que l'évaluation (bound) consiste à

évaluer les solutions d'un sous-ensemble de façon optimiste, c'est-à-dire en majorant la valeur de la meilleure solution de ce sous-ensemble.

a- Principe de la méthode

la méthode est décomposée en deux phases

La procédure de séparation

Définition

On appelle les nœuds actifs l'ensemble de nœuds de l'arbre qu'après une étape d'évaluation ils restent encore à parcourir comme étant susceptibles de contenir une solution optimale, c'est-à-dire encore à diviser.
La phase de séparation consiste à diviser le problème en un certain nombre de sous-problèmes qui ont chacun leur ensemble de solutions réalisables de telle sorte que tous ces ensembles forment un recouvrement (idéalement une partition) de l'ensemble S (espace des solutions). Ainsi, en résolvant tous les sous-problèmes et en prenant la meilleure solution trouvée, on est assuré d'avoir résolu le problème initial.
Ce principe de séparation peut être appliqué de manière récursive à chacun des sous-ensembles de solutions obtenus, et ceci tant qu'il y a des ensembles contenant plusieurs solutions. Les ensembles de solutions (et leurs sous-problèmes associés) ainsi construits ont une hiérarchie naturelle en arbre, souvent appelée arbre de recherche ou arbre de décision.

l'évaluation

L'évaluation d'un nœud de l'arbre de recherche a pour but de déterminer l'optimum de l'ensemble des solutions réalisables associé au nœud en question ou, au contraire, de prouver mathématiquement que cet ensemble ne contient pas de solution intéressante pour la résolution du problème (typiquement, qu'il n'y a pas de solution optimale). Lorsqu'un tel nœud est identifié dans l'arbre de recherche, il est donc inutile d'effectuer la séparation de son espace de solutions.

À un nœud donné, l'optimum du sous-problème peut être déterminé lorsque le sous-problème devient " suffisamment simple ". Par exemple, lorsque l'ensemble des solutions réalisable devient un singleton, le problème est effectivement simple : l'optimum est l'unique élément de l'ensemble. Dans d'autres cas, il arrive que par le jeu des séparations, on arrive à un sous-problème dans lequel les décisions " difficiles " ont été prises et qui peut ainsi être résolu en temps polynomial.
Pour déterminer qu'un ensemble de solutions réalisables ne contient pas de solution optimale, la méthode la plus générale consiste à déterminer une borne inférieure (resp une borne supérieure) pour le coût des solutions contenues dans l'ensemble s'il s'agit d'un problème de minimisation (resp de maximisation). Si on arrive à trouver une borne inférieure (resp supérieure) de coût supérieur (resp inférieure) au coût de la meilleure solution trouvée jusqu'à présent, on a alors l'assurance que le sous-ensemble ne contient pas l'optimum.

Stratégies de parcours

Durant le processus de construction, il faut décider de la meilleure stratégie à adopter pour le choix du prochain nœud à traiter. Bien que, ce choix considéré comme détaille d'implémentation, il peut affecter énormément les performances

Stratégie en largeur

On divise les nœuds dans l'ordre de leur création. Cette stratégie n'est pas utilisée, car elle peut mener à un nombre exponentiel de nœuds actifs et met très longtemps avant d'arriver aux premières solutions.

Stratégie de la meilleure évaluation

On divise le nœud de meilleure évaluation. L'inconvénient de cette stratégie est la place mémoire requise.

Stratégie de recherche en profondeur

on choisit pour prochain nœud actif l'un des fils du nœud qui vient d'être divisé. Si aucun de ces nœuds n'est actif on revient en arrière dans l'arbre.
Cette stratégie a plusieurs avantages :

- le nombre de nœuds actifs reste relativement faible et nécessite donc moins de mémoire ;
- l'évaluation d'un fils peut profiter de l'évaluation du père. En d'autres termes, si on vient d'évaluer le père, ce qu'il y a en mémoire permet parfois d'évaluer un fils beaucoup plus rapidement que si on le faisait plus tard ;
- plus on est profond dans l'arbre plus on a de chances de tomber sur une solution lors de l'évaluation et donc d'obtenir une valeur qui permet d'élaguer des nœuds.
- Stratégie du plus prioritaire : la priorité d'un nœud sur un autre peut dépendre d'autre chose que l'évaluation. Par exemple, il peut être avantageux de diviser un nœud potentiellement moins bon mais plus profond dans l'arbre.
- Stratégie mixte : Tant qu'on le peut (c'est à dire que l'espace mémoire ou le nombre de nœuds actifs est inférieur à un seuil fixé), on extrait le nœud de meilleure évaluation (éventuellement le plus profond parmi ceux de meilleure évaluation) ; mais quand on dépasse le seuil, on parcourt en profondeur les fils du nœud de meilleure évaluation.

Procédure d'évaluation

La borne inférieure au nombre chromatique, utilisée pour l'évaluation, est obtenue par un calcul de clique. En effet, il est bien connu que $\omega \leq \chi$

2.4.3 L'algorithme de séparation-évaluation

Règle de séparation pour notre algorithme

L'idée est basée sur le fait que colorier un graphe G est équivalent au coloriage de deux autres graphes, G' et G" , dérivés de G. Soit u et v deux sommets non adjacents

de G. G' est obtenu à partir de G en liant u et v par une arête, et G" est obtenu à partir de G en fusionnant v et u en un seul sommet (contraction de u et v). Un nouveau sommet est créé, donc dans G" au lieu de u et v, qui sera lié aux sommets adjacents à au moins un des sommets u ; v dans G.
Cette séparation est valide car les coloriages de G dans lesquels u et v auront des couleurs différentes sont 1-à-1 équivalents aux coloriages de G' . De la même manière, les coloriages de G dans lesquels u et v auront la même couleur sont 1-à-1 équivalents aux coloriages de G" [11]. On a donc le résultat suivant :

$$\chi = min\{\chi(g'), \chi(g'')\}$$

Description de l'algorithme

Les étapes de l'algorithme :

- Au niveau de la racine, on commence par le graphe initial G.
- A chaque nœud de l'arbre, deux sommets non adjacents sont choisis, et la séparation est faite en utilisant la règle décrite précédemment.
- A chaque nœud interne N, autre qu'une feuille, un calcul de clique est réalisé (B_{local} de G^N). Si $B_{local} \geq B_{best}$
- Le sous arbre ne sera pas construit car il ne mènera plus à une solution meilleure.
- Une feuille est atteinte quand le graphe G^l est complet.
- Une solution est alors obtenue. Si cette solution est meilleure que B_{best}, alors B_{best} est mise à jour.
- L'algorithme s'arrête quand toutes les branches sont explorées, et B_{best} est retourné comme solution optimale.

Conclusion

Dans ce chapitre, nous avons décrit les différents algorithmes de coloriage de graphe. Le coloriage de graphe a été utilisé pour résoudre un grand nombre de problèmes réels et théoriques.

Puisque le problème de coloriage est un problème $\mathcal{NP}$-complet, plusieurs tentatives sont fait pour créer des algorithmes efficaces pour résoudre le problème, nous avons rapporté en trois type de méthodes, les méthodes exactes où la solution est optimale mais le temps d'exécution avec des graphes de grand taille est énorme car leur complexité est exponentielle, donc ils sont utilisée avec des instances de petites et moyenne taille. Les méthodes métaheuristiques sont très utilisé pour des instances de grand taille (plus d'une centaine de sommets) où les méthodes exactes ne sont pas efficaces, D'autre part les métaheuristiques sont appliquées au PCG, constituent une adaptation d'une classe de méthodes génériques (algorithmes génétiques par exemple) au problème spécifique de coloriage. les méthodes heuristiques sont nombreuses et très efficaces en terme de temps d'éxecution grâce a leur complexité raisonnable, mais ils ne garantirent pas l'optimalité de la solution, mais ils restent très utilisés pour les graphes de grande taille.

Chapitre 3

L'approche branch-and-bound et ses performances

Dans ce chapitre nous allons détailler l'algorithme branch-and-bound utilisé dans [11] pour résoudre le problème d'ordonnancement, cité dans le chapitre précédant. nous enchainerons par relater l'étude expérimentale dans le but de mesurer en pratique les performances de cet algorithme, dans une dernière section nous allons discuter les résultats.

3.1 Préliminaire

Le but de l'algorithme est de trouver le nombre chromatique, pour cela on cherche des meilleurs bornes pour limiter l'espace de recherche de ce nombre. Une borne très connu est l'ordre de la clique maximale $\omega(G)$ d'un graphe G, mais cette borne reste encore difficile à calculer.

En effet le calcul de ω est aussi un problème $\mathcal{NP}$-complet, on doit donc chercher une autre borne.

Utilisant la formulation de la programmation semi-definie, Lovasz [9] a défini une nouvelle borne inférieure au nombre chromatique $\chi(G)$. Cette borne nommée thêta $\vartheta(G)$, définie par le théorème de sandwich affirmant qu'elle est comprise entre la taille de la

clique maximale ω et χ comme suit

$$\omega(\mathrm{G}) \leq \vartheta(\bar{G}) \leq \chi(\mathrm{G})$$

$\vartheta(\bar{G})$ est une borne inférieure de $\chi(G)$ meilleure que la taille de la clique maximale ω(G). D.Knuth et al. ont prouvé que cette borne peut être calculée dans un temps polynomial ce qui a permi d'avoir une bonne fonction d'évaluation.

3.1.1 La fonction Thêta

La fonction thêta se définit sur un graphe pour calculer une borne inférieure au nombre chromatique de ce graphe en un temps polynomial $\vartheta(\bar{G})$. Cette fonction possède plusieurs propriétés intéressantes et peut être caractérisée de plusieurs manières.
Avant de préciser ses propriétés et caractéristiques il est important de rappeler quelques notions de l'algèbre.

Définitions et notation

On note les matrices en majuscule et les vecteurs en miniscule.

- A est une matrice.
- a et b sont des vecteurs.

On note aussi :

- A_v est un vecteur matrice.
- λ est la valeur propre de la matrice A.
- I represente la matrice identité.

Nous avons :

Propriété 1

$$\text{a.b} = a^T\text{b}$$

$\langle$ a.b $\rangle$ est le produit des deux vecteurs a et b.

Si a.b = 0 alors a et b sont dits vecteurs orthogonaux (ou encore perpendiculaires),

Propriété 2

Pour toute matrice A on a :

$$(\mathrm{A}^T\ \mathrm{A})_{uv} = \sum_{k=1}^{n}(A^T)_{uk}A_{kv} = \sum_{k=1}^{n} A_{uk}A_{kv} = \mathrm{A}_u.A_v$$

une matrice A est dite orthogonale si A est une matrice carré et la matrice identité A^T A= I.

on peut déduire de cette formule qu'une matrice est orthogonale si et seulement si ses colonnes ont, deux à deux, un produit scalaire égale à 1 [11].

Soit un graphe G = (V ;E). La fonction thêta pondérée est une fonction à deux paramètres, notée $\vartheta(\bar{G}, \omega)$, où ω représente un étiquetage réel non négative, elle est définie comme suit :

$$\vartheta(\bar{G}, \omega) = \max\{\ \omega.x \mid x \in TH(G)\}$$

TH(G) est un l'ensemble des étiquetages réels déni comme suit : TH(G) = $\{x \geq 0 \mid \sum_{i \in v} c(a_i)x_i \leq 1$ pour toutes les représentations orthogonales a de G$\}$

La fonction thêta, notée $\vartheta(\bar{G})$ est un cas particulier de la fonction thêta pondérée $\vartheta(\bar{G}, \omega)$ pour laquelle $\omega_i = 1$.

la propriété la plus importante est :

$$\omega(\mathrm{G}) \leq \vartheta(\bar{G}) \leq \chi(\mathrm{G})$$

Il faut noter aussi que la fonction thêta est une quantité plus abstraite, spécialement quand il s'agit d'une représentation orthogonale de dimension supérieure à 3. pour un espace de 3 dimention, géométriquement $\vartheta(\bar{G})$ est le plus petit angle du cône de rotation qui contient tous les vecteurs d'une représentation orthogonale d'un graphe G sur R^3. Nous allons citer deux caractéristiques pour la fonction thêta [11] :

Caractéristique 1

Pour tout graphe G, Thêta est définie par :

$$\vartheta(G,\omega) = min_{A \in M_\omega(G)}\ \lambda_{max}(A)$$

$\vartheta(G,\omega)$ est la petite valeur parmi les valeurs propres maximales ($_{max}(A)$), prise sur toutes les matrices A de M(G). Notons que l'ensemble M(G) est un ensemble infini. donc $\vartheta(G,\omega)$ n'est pas trivial.

Caractéristique 2

Soit b une représentation orthonormée pour $\bar{G}$. on aura :

$$\vartheta(G,\omega) = \max_b \sum_{u,v} (\sqrt{\omega_u b_u}).(\sqrt{\omega_v b_v})$$

où b est pris sur toutes les représentations orthonormées de $\bar{G}$. Le nombre de représentations orthonormées de $\bar{G}$ peut être infini.
[9] a prouvé qu'on peut formuler $\vartheta(G,\omega)$ comme étant un Problème semidéfinie (SDP), grâce a la première caractéristique. ce problème peut être résolu en un temps polynomiale.
Parmi ces formulations, la SDP semble la plus simple à comprendre et calculer.

3.1.2 La programmation semi définie

La programmation semidéfinie est un sous-champ de l'Optimisation Convexe (OC), concernée par l'optimisation d'une fonction objective linéaire sur l'intersection du cône de matrices semi définie positives avec un espace affine. On peut dire aussi qu'elle est un problème d'optimisation continue [7], et elle est considérée comme une extension de la

programmation linéaire. De nombreuses méthodes ont été proposées pour la programmation linéaire et ont ensuite été généralisées pour la programmation semi-définie [7]. Un programme semi définie dit primal, sous forme standard s'écrit comme suit :

$$(PSDP)\begin{cases} Min\langle C, X\rangle \\ \quad s.cAX = b \\ X \geq 0 \end{cases}$$

$b \in \mathbf{R}$ et A est un opérateur linéaire de S_n dans $\mathbf{R}^m telque$

$$(AX)\begin{pmatrix} \langle A_1, X\rangle \\ \vdots \\ \langle A_m, X\rangle \end{pmatrix}$$

C et A_i, i = 1, . . . ,m, sont des matrices de M_n. Sans perte de généralité on peut supposer que C et les A_i sont symétriques.

Thêta sous forme d'une formulation semi définie

La fonction thêta peut être donnée par optimum d'un programme semi défini [7]. $\vartheta(G)$ est l'optimum de psd donné par :

$$Minimiser\begin{cases} t \end{cases}$$

$$Sous\_contrainte\begin{cases} Y \geq 0 \\ Y_{ij} = -1(\forall ij \in E(\bar{G})) \\ Y_{ii} = t - 1 \end{cases}$$

Aprés ces definitions, nous revenons pour décrire l'algorithme à étudier.

3.2 L'algorithme

En effet, cet algorithme prend comme entrée un graphe G et donne le nombre chromatique $\chi(G)$ comme résultat. L'algorithme basé sur le schéma de séparation et évaluation, Il construit un arbre de sous problèmes. On fait un parcours en profondeur jusqu'on arrive aux feuilles. En éliminant les branches qui nous guident vers des solutions supérieure à la meilleure solution trouvée jusqu'à lors [11].

Le pricipe de l'algorithme est le suivant :

1. On choisit deux nœuds non adjacents u et v.
2. On obtient deux branches, G^L où on ajoute une arête entre u et v, et G^R où on fait contraction de u et v.
3. B_{best} est le nombre de couleur minimal valide trouvé jusqu'à lors, au début il est initialisé au degré de nœud maximum + 1 ($\Delta(G) + 1$).
4. Pour chaque nœud non feuille, on fait le calcul de thêta $\vartheta(G)$ et on l'affecte à (B_{local}) appliqué au graphe obtenu à chaque fois. Si $B_{local} \geq B_{best}$ alors le sous arbre qui suit ne sera pas construit car il ne mènera plus à une solution meilleure.
5. Une feuille est atteinte quand il n'y a plus de sommets non adjacents dans le graphe,une solution réelle est alors atteinte. Si cette solution est meilleure que B_{best}, alors B_{best} est mise à jour.
6. L'algorithme s'arrête quand on arrive à la feuille la plus à droite donc on aurra exploré tous l'espace des solutions, et B_{best} représente la solution finale au problème.

3.3 Etude expérimentale

Dans cette partie, nous allons évaluer les performances de cet algorithme, qu'est initialement associé au problème d'ordonancement. Il se base sur un schéma à séparation et évaluation.

Pour évaluer les performances de cet algorithme nous avons :

– Implémenté un programme appelé BAB (implémenté efficacement en C).

– Utilisé un programme calculant $\vartheta(\bar{G})$.

– Plus un jeu d'essai ou des instances de graphes tirées d'un benchmark [8].

Nous avons implémenté cet algorithme B.A.B, en langage C pour pouvoir manipuler toutes les structures à bas niveau et maitriser les appels récursifs efficacement. Pour le calcul de thêta deux possibilités existent : utilisé les outils existants ou faire de développement propre.

Nous avons choisit de réutiliser les codes existants, ils sont très nombreux et notre choix c'est porté sur l'outil CSDP [7]. On peut obtenir $\vartheta(\bar{G})$ par deux méthodes :

– Soit par un appel système.

– Soit par appel aux fonctions d'une bibliothèque écrite en C.

3.3.1 L'outil CSDP

CSDP est une bibliothèque de routines qui implémente une variante de l'algorithme de résolution des programmes semidefinies de Helmberg et al [7]. Les principaux avantages de cet outil est qu'il est écrit pour être utilisé comme un appel à des routines, il est écrit en langage C pour l'efficacité, et qu'il fait un usage efficace des trous (blocs de zéros) dans les matrices de contraintes.

Un autre avantage de cet outil est que son auteur, conscient de l'importance de la fonction thêta, a développé une fonction qui calcule la valeur de thêta pour un graphe en générant le programme semi défini correspondant directement.

Cette fonctionnalité, est absente dans les autres outils de résolution des PSD. la version utilisé est la 6.1.1.

3.3.2 Description de jeu de test

Afin d'avoir une vue globale sur les performances de l'algorithme étudie, nous avons utilisé des instances de coloriage de graphe, Pour chaque instance, nous allons appliquer l'algorithme et prendre les résultats c-à-d le temps d'exécution.

Les instances choisit sont des instances crées par des communautés, de la théorie des graphe : DIMACS Challenge [8].

Ce benchmark représente des applications dans plusieurs domaines d'ingénierie et de calculs scientifiques, DIMACS Challenge est crée pour faciliter les efforts nécessaires pour tester et comparer des algorithmes et des heuristiques en fournissant un ensemble riche d'instances de graphes et d'outils d'analyse. Les instances choisies sont décrit par le tableau suivant.

3.3.3 Résultats et discussion

Nous avons implémenté l'algorithme BAB en C, sur une machine tournant sous linux (JOSSLINUX). notre machine a des caractères modeste suivantes :
NETBOOK
CPU : Intel Atom CPU 720, 1,60GHZ 1,60GHZ.
RAM : 1G.
Nous avons aussi tester deux parcours en profondeur et largeur mais ou le nombre de'arête généralement inférieur a n^2.

Instance	N	M	$\chi(G)$	$\vartheta(\bar{G})$	Td	Tg
1-FullIns$_3$	30	100	4	3	1,17 s	2 s
2-FullIns$_3$	52	201	5	4	69 s	71 s
2-Insertion$_3$	37	72	4	2	0,07 s	0,15 s
3-Insertion$_3$	56	110	4	2	18 s	19 s
css1	15	11	2	2	0,09 s	0,19 s
css12	17	13	3	3	0,06 s	0,13 s
css11	15	11	3	2	0,09 s	0,18 s
css2	32	22	2	2	0,28 s	0,54 s
css3	27	23	2	2	0,17 s	0,39 s
css5	9	6	2	2	0,03 s	0,09 s
css6	12	4	2	2	0,03 s	0,06 s
wss1	44	56	2	2	0,24 s	1,26 s
wss2	32	24	2	2	0,16 s	0,48 s
wss3	11	6	2	2	0,03 s	0,04 s
wmt22	31	40	2	2	0,16 s	0,98 s
jac1	19	13	2	2	0,14 s	0,24 s
queen5-5	25	160	5	5	0,25 s	2,65 s
queen6$_6$	36	290	7	6	24,7 s	25,1 s
queen7$_7$	49	476	7	7	49 s	50 s
queen8$_8$	64	728	9	8	64 s	66 s
myciel3	11	20	4	2	0,16 s	0,26 s
myciel4	23	71	5	2	29 s	30 s
myciel5	47	236	6	2	6,33 s	7,03 s

TABLE 3.1 – Résultat d'exécution

Dans le table 3.1, nous présentons les résultats de nos expériences.
les colonnes sont définit comme suit :

– Instance : le nom de l'instance.
– N : nombre de nœud.
– M : nombre d'arête.
– $\chi(G)$: le nombre chromatique.
– $\vartheta(\bar{G})$ la valeur de thêta.
– Td : le temps d'exécution pour un parcours en profondeur a droite.
– Tg : le temps d'exécution pour un parcours en profondeur a gauche.

La première constatation que l'on peux faire sur ces résultats est le fait que la valeur $\vartheta(G)$ est très proche du nombre $\chi(G)$ Les temps d'exécutions sont très acceptables. Cependant, ou les caractéristique modeste de notre machine nous avons pas pu dépassé des instances de plus de 80 nœuds.

Conclusion

Le problème de coloriage de graphes reste toujours prend une grande importance, vue l'utilisation de ce modèle pour résoudre de nombreux problèmes.
Le manque des méthodes exactes exige de chercher de nouvelles méthodes. Dans ce travail, nous avons étudie un algorithme exacte utilisant une métaheuristique, Cet algorithme est conçu initialement pour donner une solution à un problème d'ordonnancement des tâches où le problème est modélise comme un problème de coloriage de graphe.
Nous avons mesuré les performances de cet algorithme qui se basé sur un schéma à séparation et évaluation branch-and-bound [4] précisément pour le problème de coloriage. Les résultats de l'expérimentation ont montré que l'algorithme donne des bonnes résultats, en terme de temps d'exécution pour des instances allant jusqu'a 80 nœuds.

Bibliographie

[1] NEHAR Attia. Algorithmes exactes de coloriage de graphes pour l'ordonnancement, 2009.

[2] BAPTISTE AUTIN. Les métaheuristiques en optimisation combinatoire, 2006.

[3] Daniel Brélaz. New methods to color the vertices of a graph. *Commun. ACM*, avril 1979.

[4] M. Caramia and Dell'Olmo. Vertex coloring by multistage branch-and-bound. in computational symposium on graph coloring and its generalizations (color02). *Ithaca, N-Y*, 2002.

[5] P. Diaz, I. M. et Zabala. A branch-and-cut algorithm for graph coloring,in computational symposium on graph coloring and its generalizations (color02). *Ithaca, N-Y*, 2002.

[6] Christophe DUMEUNIER. Problèmes de coloriage de graphes, 2008-2009.

[7] ALABBOUD Hassan. *La programmation semi-définie combinée et comparée avec dautres probèmes doptimisation.* PhD thesis, lUniversit´e du Havre, decembre 2007.

[8] D.S. Johnson and M. Trick. Cliques, coloring, and satisfiability : Second DIMACS implementation challenge. *American Mathematical Society*, 1996.

[9] Laszlo Lovasz. Geometric representations of graphs. 2007.

[10] Mehrotra and Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 1996.

[11] Attia Nehar, Hadda Cherroun, and Younes Guellouma. Algorithme exacte de coloriage de graphes pour lordonnancement. *Colloque sur l'optimisation et les Systèmes d'information COSI2009*, mai 2012.

[12] Daniel Cosmin PORUMBE. *Algorithmes Heuristiques et Techniques d'Apprentissage Applications au Problème de Coloration de Graphe.* PhD thesis, Université angers, 2009.

[13] M B Powell and D J A Welsh. Upper bound for the chromatic number of a graph and its applications to timetabling problems. *The Computer Journal*, 1967.

[14] Michel Rigo. *Théorie des graphes.* université de liège, cours, 2009/2010.

[15] Nicolas Schabanel. *Algorithmes d'approximation pour les télécommunications sans fil : Ordonnancement pour la dissémination de données et Allocation statique de fréquences.* PhD thesis, l'École Normale Supérieure de Lyon, janvier 2000.

[16] Sewell. An improved algorithm for exact graph coloring. 1993.