**N° d'ordre : 03/2018-D/MT**

**REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE**
**Ministère de l'Enseignement Supérieur et de la Recherche Scientifique**
**Université des Sciences et de la Technologie Houari Boumediène**

## Faculté de Mathématiques

جامعة هواري بومدين
للعلوم والتكنولوجيا
U S T H B

**THESE**

Présentée pour l'obtention du **grade** de **DOCTEUR EN SCIENCES**

**En** : MATHÉMATIQUES

**Spécialité** : Recherche Opérationnelle

**Par** : KOUIDER Ahmed

**Sujet**

# Coloration de Graphes et ses Applications aux Problèmes d'Ordonnancement

Soutenue publiquement, le 17/03/2018, devant le jury composé de :

| | | |
|---|---|---|
| M. BERRACHEDI Abdelhafid, | Professeur à l'USTHB, | Président |
| M. AIT HADDADENE Hacène, | Professeur à l'USTHB, | Directeur de thèse |
| M. OULAMRA Ammar, | Professeur à l'Université de Lorraine, | Co-Directeur de thèse |
| M. AIT ZAI Abdelhakim, | Maître de Conférence /A à l'USTHB, | Examinateur |
| M. CHELLALI Mustapha, | Professeur à l'USDBlida1, | Examinateur |
| M. SADI Bachir, | Maître de Conférence /A à l'UMMTO, | Examinateur |
| M. AIDER Méziane, | Professeur à l'USTHB, | Invité |

# Remerciements

Je tiens en premier lieu à exprimer ma reconnaissance à mon directeur de thèse Monsieur AIT HADDADENE Hacène, Professeur à l'Université des Sciences et de la Technologie Houari Boumediene (USTHB), qui m'a permis d'effectuer cette thèse dans les meilleures conditions. Je le remercie vivement pour son aide, ses orientations, sa disponibilité et ses multiples conseils judicieux.

J'adresse mes remerciements à mon codirecteur de thèse Monsieur OULAMARA Ammar, Professeur à l'université de Lorraine, pour la confiance qu'il m'a accordée en acceptant de codiriger ce travail et de le suivre malgré ses nombreuses charges. Je le remercie pour son accueil à l'université de Lorraine durant mon séjour scientifique.

Mes vifs remerciements s'adressent à l'ensemble des membres de mon jury de thèse. Je remercie Monsieur BERRACHEDI Abdelhafid, Professeur à l'USTHB, pour l'honneur qu'il m'a fait en acceptant de présider le jury.
Je tiens à exprimer mes profonds remerciements à Monsieur AIDER Méziane, Professeur à l'USTHB, Monsieur AIT ZAI Abdelhakim, Maître de conférences (A) à l'USTHB, Monsieur CHELLALI Mustapha, Professeur à l'USDBlida1 et Monsieur SADI Bachir, Maître de conférences (A) à l'UMMTO, pour avoir accepté d'examiner ce travail et d'être membre du jury.

Un grand merci également à tous ceux qui ont pu m'aider d'une manière ou d'une autre. Je remercie particulièrement mes collègues de la Division Productique et Robotique pour leur soutien et leur amitié, en particulier Madame OURARI Samia, Maître de Recherche au CDTA, pour ses commentaires, ses conseils et son aide. Je remercie Monsieur BOUZOUIA Brahim, Directeur de recherche au CDTA et responsable de l'équipe Systèmes Robotisés de Production à laquelle j'appartiens, pour sa confiance.

Je tiens à remercier plus personnellement mon père, ma mère, mon frère et toute ma famille pour leur soutien et leur compréhension. Je remercie en particulier, mon épouse ainsi que mes enfants Mahdi, Sara et Abderrahmane pour leur patience, pour la joie qu'ils m'apportent et pour la motivation qu'ils me donnent.

# Table des matières

# Liste des figures

# Liste des tableaux

# Liste des algorithmes

# Introduction générale

L'idée de partitionner un ensemble d'objets en classes suivant certaines règles fait partie d'un processus fondamental en mathématiques. Sa concrétisation nécessite un ensemble de règles conceptuellement simples permettant de décider pour toute paire d'objets s'ils sont autorisés ou pas à appartenir à une même classe. Formellement, il est évident de modéliser ce problème par un graphe, où les objets considérés sont les sommets de ce graphe et deux sommets sont adjacents si et seulement s'ils sont contraints à ne pas appartenir à une même classe. Ainsi, la coloration de graphes traite exactement ce type de problèmes de décision en optimisant un ou plusieurs critères.

L'une des premières sources d'inspiration de la coloration de graphes a été le problème des 4 couleurs, qui est resté ouvert pendant presque un siècle avant d'être démontré par (Appel and Haken, 1976). Depuis lors, les problèmes de coloration de graphes intéressent beaucoup de chercheurs et nombreux travaux sont publiés dans la littérature avec de pertinents résultats mathématiques. La coloration de graphes permet de modéliser divers problèmes pratiques, notamment ceux qui sont issus des domaines technologiques et connaissant un fort développement, à savoir les réseaux de communications et l'ordonnancement des systèmes de production de biens et de services.

L'ordonnancement est une fonction importante rencontrée dans de nombreux domaines, et qui consiste à allouer des ressources aux tâches sur des périodes de temps à déterminer, en optimisant un ou plusieurs critères. Pour ce type de problèmes bien pratique, les objets peuvent être des tâches à partitionner en classes suivant certaines règles imposées par les contraintes du problème, et chaque classe regroupe un sous ensemble de tâches non soumises aux conflits imposés par ces contraintes. Ainsi la coloration de graphes permet de modéliser divers problèmes d'ordonnancement, et lorsque certaines tâches sont soumises à des exigences d'incompatibilité, elles sont alors assignées à des classes différentes.

C'est dans cette optique que les travaux de cette thèse sont réalisés, et sont décrits en quatre chapitres organisés comme suit :

**Chapitre 1 : Notions de base**

Le premier chapitre rappelle en premier quelques définitions de la théorie des graphes nécessaires à la compréhension de cette thèse, en particulier celles liées aux problèmes de coloration des graphes (non orientés et mixtes) qui permettent de modéliser certains types de

problèmes d'ordonnancement. Ensuite, il présente des généralités sur l'ordonnancement, en précisant les principaux éléments nécessaires à la connaissance de ce domaine et une caractérisation de certains problèmes d'ordonnancement spécifiques. Puis enfin, un aperçu rapide de quelques notions élémentaires de la théorie de complexité.

**Chapitre 2 : Problématique et état de l'art**

Ce chapitre introduit en première partie les problèmes d'ordonnancement auxquels nous nous intéressons dans ce travail, en précisant les différentes contraintes, les hypothèses de travail ainsi que les deux objectifs considérés. Puis, nous décrivons la réduction des deux problèmes d'ordonnancement job shop avec des opérations de durée unitaire ($J|p_{ij} = 1|C_{max}$) et ($J|p_{ij} = 1|\sum C_i$) aux problèmes de coloration de graphes mixtes, et nous présentons quelques cas pratiques tirés de la littérature où ils sont appliqués. Dans la deuxième partie, nous présentons une étude de la complexité de ces problèmes d'ordonnancement ainsi que celle de quelques autres problèmes d'ordonnancement qui leur sont particuliers ou leur sont plus généraux. Enfin, la troisième partie du chapitre passe en revue un état de l'art sur les principaux travaux de recherche traitant des problèmes d'ordonnancement avec des durées unitaires et des problèmes de coloration de graphes mixtes.

**Chapitre 3 : Méthodes de résolution**

La contribution présentée dans ce chapitre est basée sur la réduction des deux problèmes d'ordonnancement étudiés aux deux problèmes de coloration de graphes mixtes, où le premier problème s'intéresse au nombre chromatique ($\chi$) et le second à la minimisation de la somme des couleurs des extrémités terminales des chemins de longueur maximale ($\sigma$). Une borne inférieure et une borne supérieure sont proposées pour chacun des critères à optimiser des deux problèmes de coloration. Ensuite, pour la résolution exacte de chacun des deux problèmes, un modèle mathématique caractérisé par l'indexation de la couleur au niveau des variables décisionnelles est proposé.
Nous avons aussi proposé une seconde méthode exacte basée sur un algorithme de séparation et évaluation adaptée à la résolution des deux problèmes de coloration de graphes mixtes. En se basant sur la force et les limites des principales composantes de ce type de méthodes que nous avons spécifiées, nous décrivons une adaptation de ces composantes pour la résolution des deux problèmes de coloration étudiés. Enfin, la dernière partie de ce chapitre s'intéresse à la résolution approchée, où nous exposons le principe de fonctionnement de l'algorithme tabu search et décrivons par la suite son adaptation pour la résolution des deux problèmes de coloration.

**Chapitre 4 : Expérimentations et résultats**

Le dernier chapitre du manuscrit est consacré à l'évaluation et l'analyse des performances des bornes et des méthodes de résolution proposées pour les problèmes de coloration de graphes mixtes. L'environnement et les outils utilisés dans les différentes expérimentations

réalisées sont présenté. Ensuite, les différentes instances de problèmes utilisées pour la validation des méthodes et des bornes proposées sont décrites. En s'appuyant sur quelques travaux de la littérature, nous donnons une description de quelques critères de classification des instances difficiles caractérisant le type de problèmes étudiés. Notre but est de montrer que la difficulté des instances ne se limite pas seulement à leur taille (i.e. l'ordre du graphe modélisant le problème d'ordonnancement), mais plutôt à la structure de ces instances qui est définie par une relation entre l'ordre du graphe, le nombre de cliques maximales et le nombre de chemins de longueur maximale.

Avant de procéder à l'évaluation de la performance des méthodes basées sur la modélisation mathématique, nous avons effectué une étude expérimentale pour montrer empiriquement que l'efficacité de ces méthodes est étroitement dépendante de l'efficacité du paramétrage du solveur adopté.

Les résultats expérimentaux présentées dans ce chapitre sont scindés en deux principales parties selon le critère utilisé. Dans les deux cas, les mêmes instances (benchmarks modifiés et instances générées de manière aléatoire) et les mêmes indicateurs de performance sont utilisés. La première partie décrit les expérimentations utilisées pour le problème de coloration de graphes mixtes avec le critère du nombre chromatique ($\chi$), et la seconde partie décrit les expérimentations utilisées pour la coloration de graphes mixtes optimisant la somme des couleurs des extrémités terminales des chemins de longueur maximale ($\sigma$). Dans chacune des parties, nous validons les deux bornes inférieure et supérieure proposées, puis nous analysons la performance des méthodes exactes et de la méthode approchée.

Enfin nous présentons une conclusion générale où nous résumons les résultats obtenus et les perspectives envisagées.

# Chapitre 1

# Concepts de base

## Sommaire

## 1.1 Introduction

Ce chapitre constitue un rappel des notions et terminologies utilisées dans l'ensemble de ce manuscrit. Il rappelle quelques définitions de base de la théorie des graphes dans une première section, puis quelques notions élémentaires relatives aux problèmes d'ordonnancement dans une deuxième section. Un petit aperçu sur la théorie de la complexité algorithmique est donné en dernière section.

## 1.2 Graphes et coloration de graphes

Dans cette section, nous nous limitons à introduire quelques définitions et notations relatives aux graphes et à la coloration de graphes utilisés dans la suite de ce manuscrit. Pour plus d'information concernant ces notions, le lecteur peut se référer aux ouvrages suivants : (Bondy and Murty, 2008), (Golumb, 2004) et (Diestel, 2000).

### 1.2.1 Notions de base et notations

Un graphe non-orienté (resp. orienté) est un couple, (V(G), E(G)) (resp. (V(G), A(G))) composé d'un ensemble non vide de sommets V(G) et d'un ensemble d'arêtes E(G) (resp. d'arcs A(G)), que nous notons simplement par la suite V, E et A pour éviter toute d'ambiguïtés.

Par convention, le nombre de sommets de G est noté $n$. De même, le nombre d'arcs (ou d'arêtes, selon l'orientation du graphe) contenues dans un graphe G est noté $m$. Mais par souci d'analogie avec le problème d'ordonnancement étudié, ces deux notions sont exceptionnellement modifiées dans le chapitre suivant.

Dans un graphe non orienté, une arête reliant le sommet $v_i \in$ V et le sommet $v_j \in$ V peut être notée $v_i v_j \in$ E ou $[v_i, v_j] \in$ E, où $v_i$ et $v_j$ sont ses extrémités. Alors que dans un graphe orienté, un arc reliant le sommet $v_i \in$ V et le sommet $v_j \in$ V peut être noté $v_i v_j \in$ A ou $(v_i, v_j) \in$ A, où $v_i$ est l'extrémité initiale et $v_j$ est l'extrémité terminale.

Une arête ayant pour extrémités le même sommet est appelée boucle, et les arêtes possédant la même paire d'extrémités sont dites arêtes multiples.
Un graphe G est dit simple s'il ne possède ni boucles ni arêtes multiples, et un graphe contenant un sommet unique s'appelle un singleton.

#### Chaine, cycle, chemin et circuit

Dans un graphe G = (V, E), une chaîne entre un sommet $v_i \in$ V et un sommet $v_j \in$ V est une séquence d'arêtes consécutives ayant les sommets $v_i$ et $v_j$ pour extrémités, et la longueur de la chaîne est le nombre d'arêtes qui la compose. Une chaîne dont ses extrémités coïncident est appelée cycle.

Dans un graphe orienté G = (V, A), un chemin allant d'un sommet $v_i \in V$ à un sommet $v_j \in V$ est une séquence d'arcs consécutifs ayant le sommet $v_i$ comme extrémité initiale et $v_j$ comme extrémité terminale, et la longueur d'un chemin est le nombre d'arcs qui le compose. Un chemin dont ses extrémités initiale et terminale coïncident est appelé circuit.
Un chemin (ou une chaîne) est dit élémentaire s'il ne passe pas deux fois par le même sommet.
Un graphe G est connexe, si pour toute paire de sommets $(v_i, v_j)$ dans $V^2$, il existe une chaîne reliant $v_i$ à $v_j$.

**Union et intersection de graphes**

Deux graphes $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$ sont dits sommets-disjoints (ou simplement disjoints) si $V_1 \cap V_2 = \emptyset$. L'union disjointe de $G_1$ et $G_2$ est le graphe ayant pour ensemble de sommets $V_1 \cup V_2$ et pour ensemble d'arêtes $E_1 \cup E_2$. L'intersection de $G_1$ et $G_2$ est le graphe ayant pour ensemble de sommets $V_1 \cap V_2$ et pour ensemble d'arêtes $E_1 \cap E_2$.

**Voisinage et degré**

On appelle voisinage d'un sommet $v$, noté $N(v)$, l'ensemble des sommets qui sont adjacents à $v$. Un sommet $v_i \in V$ est adjacent à un sommet $v_j \in V$, si $v_i$ et $v_j$ sont reliés par une arête. Le degré $d(v)$ du sommet $v$ dénote le cardinal du voisinage de $v$ ($|N(v)|$). Le degré maximum du graphe G est égal à $\max_{v \in V} d(v)$, et est noté $\Delta(G)$. Un sommet isolé est un sommet dont le voisinage est l'ensemble vide.

**Graphe complet, clique et ensemble indépendant**

Un graphe non-orienté G = (V, E) est complet si et seulement si chaque paire de sommets est connectée par une arête. On note $K_n$ le graphe complet d'ordre $n$.
Une clique de G, notée K, est un ensemble de sommets adjacents deux à deux. La taille d'une clique maximum $\omega(G)$ est égale au nombre de ses sommets, et la partition minimum en cliques est noté $\theta(G)$.
Un stable de G, noté S, est un ensemble de sommets deux à deux non adjacents. La taille d'un stable maximum $\alpha(G)$ est égale au nombre de ses sommets.

**Graphe complémentaire, sous graphe et graphe partiel**

Considérons un graphe non orienté $G = (V, E)$. Le graphe complémentaire de G est $\overline{G} = (V, \overline{E})$ tel que pour toute paire de sommets $(v_i, v_j)$ de $V^2$, $v_i v_j \in \overline{E}$ si et seulement si $v_i v_j \notin E$.
Un graphe partiel $G_1 = (V, E_1)$ de G est un graphe tel que $E_1 \subset E$, et est obtenu en supprimant des arêtes au graphe G.
Un sous graphe de G induit par $V_2$ ($V_2 \subset V$) est le graphe $G_2 = (V_2, E_2)$ tel que $E_2 = \{v_i v_j \in E, v_i \in V_2, v_j \in V_2\}$. Aussi, $G_3 = (V_3, E_3)$ est un sous-graphe partiel de G si $G_3$ est un graphe partiel du sous-graphe de G induit par $V_3$, ainsi $V_3 \subset V$ et $E_3 \subset E$ est un sous-ensemble de l'ensemble des arêtes de G d'extrémités dans $V_3$.

### 1.2.2 Coloration de graphes

Une coloration des sommets d'un graphe non orienté G=(V, E) est une application $c$ : V → S de l'ensemble des sommets V de G dans l'ensemble discret S de couleurs, qui associe à chaque sommet de V une couleur de S de telle sorte que deux sommets adjacents dans G reçoivent des couleurs différentes, i.e. si $uv \in$ E, alors $c(u) \neq c(v)$. Le problème de coloration de graphes cherche à trouver une application $c$ minimisant la cardinalité de l'ensemble S.

A partir de la cardinalité de l'ensemble S (|S|), le nombre chromatique d'un graphe ($\chi$(G)) est défini par la plus petite valeur de |S| pour qu'une coloration existe.

Bien que la coloration de graphe est l'un des 21 problèmes montrés NP-complet dans le célèbre papier de Karp (Karp, 1972), une importante classe de graphes pour lesquels le problème de coloration peut être résolu en un temps polynomial existe et est appelée classe des graphes parfaits. Cette classe de graphes est introduite en 1960 par Berge (1960), et est devenue depuis lors un objet d'étude important en mathématiques et a trouvé application dans divers domaines.
Il existe plusieurs caractérisations des graphes parfaits, parmi lesquels nous citons : *un graphe G est dit parfait si tout sous-graphe induit H de G vérifie* $\chi(H) = \omega(H)$. Pour une synthèse complète sur les graphes parfaits, voir les références : (Lovasz, 1972) et (Berge and Chvatal, 1984).

Dans le cas général, déterminer le nombre chromatique ($\chi$) d'un graphe quelconque est un problème NP-difficile. Pour cette raison, plusieurs travaux de la littérature se sont intéressés à déterminer $\chi$ pour des classes particulières de graphes, ou bien à proposer des bornes inférieures ou supérieures pour $\chi$ en fonction d'autres paramètres de graphes.

Parmi les bornes supérieures proposées dans la littérature, nous citons celle de Brooks (1941) qui établit une relation entre le nombre chromatique d'un graphe non orienté et son degré maximum :

**Théorème 1.** *(Brooks, 1941)*
*Pour tout graphe* G, $\chi(G) \leq \Delta(G) + 1$. *De plus,* $\chi(G) = \Delta + 1$ *si et seulement si,* $\Delta(G) \neq 2$ *et l'une des composantes connexes de* G *est un graphe complet d'ordre* $\Delta(G) + 1$*, ou bien* $\Delta(G) = 2$ *et l'une des composantes connexes de* G *est un cycle impair.*

Un autre résultat trivial bien connu est que la taille d'une clique d'un graphe G constitue une borne inférieure pour le nombre chromatique $\chi$. En effet, si un graphe G contient une clique de taille $\omega$, alors il faut au moins $\omega$ couleurs pour le colorer. Ainsi, tout graphe G vérifie la relation : $\omega \leq \chi$.

Parmi les nombreux résultats relatifs au nombre chromatique nous pouvons citer le célèbre résultat de Appel et Haken obtenu en 1976.

**Théorème 2.** *(Appel and Haken, 1976)*
*Tous graphe planaire est 4-coloriable.*

Il est bien connu que la coloration de graphes non orientés permet de modéliser plusieurs problèmes réels. Nous citons comme exemple les problèmes d'ordonnancement, où les tâches à ordonnancer sont soumises à des exigences d'incompatibilité (i.e. certaines tâches ne peuvent être exécutées dans une même période de temps). Toutefois, ce type de coloration est limité et ne peut prendre en charge les contraintes de précédence entre certaines tâches imposées par la nature de certains problèmes. Ainsi, pour modéliser des problèmes d'ordonnancement avec les deux types de contraintes, la coloration de graphes mixtes peut être utilisée, et la sous-section suivante est dédiée à son introduction.

### 1.2.3 Coloration de graphes mixtes

Un graphe mixte G = (V, E, A) est défini par un ensemble non vide de sommets, un ensemble d'arêtes (E) et un ensemble d'arcs (A). Dans ce type de graphes, une arête joignant deux sommets $v_i$ et $v_j$ est notée $[v_i, v_j]$, alors qu'un arc d'extrémité initiale $v_p$ et d'extrémité terminale $v_q$ est noté $(v_p, v_q)$. Ainsi, une contrainte de précédence imposant une tâche $p$ à être traitée avant une autre tâche $q$ est représentée par un arc $(v_p, v_q)$, où les sommets $v_p$ et $v_q$ représentent respectivement les tâches $p$ et $q$.

Une coloration des sommets d'un graphe mixte G=(V, E, A) est une application $\Psi : V \rightarrow S$ de l'ensemble des sommets V dans l'ensemble discret S de couleurs telle que pour tout $[v_i, v_j] \in E$, $\Psi(v_i) \neq \Psi(v_j)$ et pour tout $(v_p, v_q) \in A$, $\Psi(v_p) < \Psi(v_q)$. Généralement, le but de la coloration d'un graphe mixte est de trouver une application $\Psi$ minimisant la cardinalité de l'ensemble S, où la plus petite cardinalité de cet ensemble S est appelée le nombre chromatique du graphe mixte et est noté $\chi(G)$.

Notons que pour qu'une telle coloration existe, il faut que le graphe mixte G soit sans circuit.

## 1.3 Ordonnancement

Dans la littérature nous trouvons plusieurs ouvrages dédiés à l'ordonnancement, nous citons entre autres (Carlier and Chrétienne, 1988), (Esquirol and Lopez, 1999), et (Pinedo, 2008). De ces ouvrages, plusieurs définitions de l'ordonnancement y sont données, nous retenons celle proposée dans (Pinedo, 2008) : *L'ordonnancement est un processus décisionnel qui est utilisé régulièrement dans de nombreuses industries de production de biens et de services. Il traite l'allocation des ressources aux tâches sur des périodes de temps à déterminer, et son but consiste à optimiser un ou plusieurs objectifs.*

Les ressources et les tâches d'une organisation peuvent prendre différentes formes. Par exemple, les ressources peuvent être des machines dans un atelier, des pistes dans un aéroport, des équipes sur un chantier de construction, etc. Alors que les tâches peuvent être des opérations dans un processus de production, des décollages et des atterrissages dans un aéroport, des étapes d'un projet de construction dans un chantier etc. Chaque tâche peut avoir un certain niveau de priorité, une date de disponibilité et une date d'échéance. Les objectifs peuvent aussi prendre différentes formes. Par exemple, un objectif peut correspondre à la minimisation de la durée de réalisation de toutes les tâches, à la minimisation de la somme des dates de fin d'exécution de tâches ou à la minimisation du nombre de tâches accomplies après leurs dates d'échéance respectives, etc.

L'ordonnancement, en tant que processus décisionnel, joue un rôle majeur dans la plupart des systèmes de production de biens ainsi que dans la plupart des milieux de transport et de distribution. Nous pouvons trouver des exemples détaillants et illustrant le rôle de l'ordonnancement dans un certain nombre d'environnements réels dans l'ouvrage (Pinedo, 2008), où il a cité à titre d'exemple la fabrication des semi-conducteurs et les affectations de portes (Gates) dans un aéroport.

### 1.3.1 Notions fondamentales d'un problème d'ordonnancement

De la définition du problème d'ordonnancement nous retenons quatre notions fondamentales à décrire dans la sous-section suivante, à savoir, les tâches, les ressources, les contraintes et les objectifs à optimiser. Un travail (Job) est composé soit d'une tâche ou de plusieurs tâches, le nombre de jobs et le nombre de ressources sont supposés être finis.

#### 1.3.1.1 Tâches

Lorsqu'un job comporte une seule tâche, alors cette dernière est généralement caractérisée par quatre paramètres : sa durée opératoire $p_i$ (processing time), sa date de disponibilité $r_i$ (release time), sa date d'échéance $d_i$ (due date), et son poids $w_i$ (weight). Mais lorsqu'un job comporte plus d'une tâche, alors une tâche d'un job n'est caractérisée que par sa durée opératoire et les trois autres paramètres caractérisent le job.

Dans certains problèmes d'ordonnancement, les tâches peuvent être préemptives, et peuvent être exécutées par morceaux par une ou plusieurs ressources. Dans le cas contraire elles sont dites non préemptives.

#### 1.3.1.2 Ressources

Une ressource est un moyen destiné à être utilisé pour la réalisation d'au moins une tâche. Différents types de ressources peuvent être distinguées :

- *Ressource renouvelable* : il s'agit d'une ressource pouvant redevenir disponible en même quantité après avoir été allouée à une tâche, c'est le cas, par exemple des machines d'usinage, de robots de manutention, etc.
- *Ressource consommable* : une ressource est dite consommable si sa disponibilité décroît après avoir été allouée à une tâche (la matière première, l'énergie).
- *Ressource disjonctive* : une ressource est dite disjonctive (non partageable), si elle ne peut être affectée qu'à une seule tâche à la fois (robot manipulateur,...).
- *Ressource cumulative* : une ressource est dite cumulative (partageable), si elle peut être utilisée par plusieurs tâches simultanément.

#### 1.3.1.3 Les contraintes

Les contraintes des problèmes d'ordonnancement expriment des restrictions sur les valeurs que peuvent prendre simultanément les variables de décision. Ces contraintes sont généralement distinguées en deux types, à savoir les contraintes temporelles et les contraintes de ressources.

- *Contraintes temporelles* : ces contraintes représentent la limitation des valeurs possibles des dates de début et/ou de fin des jobs. Un job $J_i$ ne peut commencer son exécution qu'avant sa date de disponibilité $r_i$, et il est souhaité qu'il achève son exécution avant sa date d'échéance $d_i$. Dans certains cas, les tâches constituantes un job sont souvent liées entre elles par des relations d'antériorités, telle que les contraintes de gammes qui imposent le respect d'un positionnement relatif entre tâches.
- *Contraintes de ressources* : d'après la nature disjonctive ou cumulative des ressources, deux types de contraintes de ressources peuvent être distingués : pour une ressource disjonctive, ces contraintes permettent de spécifier que deux tâches ne peuvent être exécutées en même temps sur la même ressource. Alors que pour une ressource cumulative, ces contraintes permettent d'assurer que la somme des besoins simultanés des tâches pour une ressource ne peut être supérieure à la capacité de cette dernière.

#### 1.3.1.4 Les objectifs

Outre les tâches et les ressources, les problèmes d'ordonnancement sont également caractérisés par le ou les objectifs à optimiser. Les plus étudiés des objectifs sont souvent en fonction des dates de fin d'exécution des jobs, qui dépendent bien sûr de l'ordonnancement. La date de fin d'exécution d'un job $J_i$, notée $C_i$, est le temps pour lequel le job quite le système.

En se basant sur les dates de fin d'exécution des jobs, les indicateurs de performance suivants sont définis :

- le retard algébrique (lateness) qui dénote l'écart de la date de fin d'exécution $C_i$ d'un job $J_i$ par rapport à son délai souhaité $d_i$. Il est exprimé par $L_i = C_i - d_i$.
- le retard absolu (tardiness) dont la valeur est toujours positive est donné par l'expression, $T_i$=max(0, $C_i$ - $d_i$).
- l'indicateur de retard (unit penalty) défini par $U_i = \begin{cases} 1 & C_i > d_i \\ 0 & \text{sinon} \end{cases}$

Les critères usuels de performance d'un ordonnancement sont alors :

- la durée totale de l'ordonnancement (makespan) $C_{max} = \max_{i=1..n} C_i$.
- la plus grande durée de séjour $F_{max} = \max_{i=1..n} F_i$, où $F_i = C_i - r_i$.
- le plus grand retard algébrique $L_{max} = \max_{i=1..n} L_i$.
- le plus grand retard absolu $T_{max} = \max_{i=1..n} T_i$.
- la moyenne des durées de séjour $\overline{F} = \frac{1}{n} \sum_{i=1}^{i=n} F_i$.
- la moyenne des retards algébriques $\overline{L} = \frac{1}{n} \sum_{i=1}^{i=n} L_i$.
- la moyenne des retards absolus $\overline{T} = \frac{1}{n} \sum_{i=1}^{i=n} T_i$.
- le nombre pondéré de jobs en retard $N_T = \sum_{i=1}^{i=n} w_i U_i$.
- la somme pondérée des dates de fin d'exécution $\sum_{i=1}^{i=n} w_i C_i$.
- la somme pondérée des retards absolus $\sum_{i=1}^{i=n} w_i T_i$.

### 1.3.2 Les problèmes d'ordonnancement selon la structure de l'atelier

Dans les problèmes d'ordonnancement d'atelier, une ressource correspond à une machine et une tâche correspond à une opération élémentaire à réaliser sur une machine. Nous rappelons que les problèmes d'ordonnancement d'ateliers sont scindés en deux catégories selon le nombre d'opérations nécessaires à la réalisation de chaque job. La première catégorie regroupe les problèmes pour lesquels chaque job comporte une seule opération, la seconde catégorie regroupe les problèmes pour lesquels chaque job requiert plus d'une opération.

La première catégorie se subdivise en plusieurs types de problèmes selon la configuration des machines considérées :

- *Machine unique* : tous les jobs sont à exécuter sur la même machine
- *Machines dédiées* : plusieurs machines, chacune étant spécialisée pour l'exécution de certaines opérations.
- *Machines parallèles* : plusieurs machines, et tous les jobs peuvent être exécutés par ces machines.

Concernant les machines parallèles, on distingue généralement trois types différents de machines en fonction de leurs vitesses :

- *Machines parallèles identiques (Pm)* : toutes les machines présentent la même vitesse d'exécution quel que soit le job.
- *Machines uniformes* : la vitesse d'une machine diffère de la vitesse d'une autre machine par un coefficient de proportionnalité.
- *Machines indépendantes* : chaque machine présente une vitesse spécifique pour chaque job.

Les problèmes de la deuxième catégorie nécessitent le passage de chaque job sur deux ou plusieurs machines dédiées. Ces problèmes sont généralement constitués de $m$ machines qui doivent réaliser $n$ jobs. Un job correspond à une liste d'opérations élémentaires nécessaires à sa réalisation. Les opérations élémentaires de fabrication sont supposées non préemptives et les machines de production sont des ressources renouvelables.

Trois principales sous-classes de problèmes sont alors différenciées, selon le mode de passage des opérations sur les différentes machines, à savoir : Flow shop, Job shop et Open shop.

- *Atelier à cheminement unique (Flow shop)* : dans ce cas, tous les jobs utilisent les mêmes machines selon la même séquence. Autrement dit, l'atelier dispose de $m$ machines en série, et tous les jobs doivent passer dans le même ordre sur les différentes machines (cas d'une chaîne de fabrication).
- *Atelier à cheminement multiple (Job shop)* : le problème du job shop se différencie du flow shop par un élément essentiel : le cheminement des jobs n'est pas unidirectionnel. Tout job est une succession ordonnée d'opérations, et chaque job ou famille de jobs possède une gamme opératoire spécifique.
- *Atelier à cheminement libre (Open shop)* : dans le cas d'un atelier du type open shop, les opérations de chaque job peuvent être exécutées dans n'importe quel ordre (les gammes des produits sont libres). Il n'y a donc pas de contrainte de précédence entre deux opérations d'un même job.

### 1.3.3 Classification des problèmes d'ordonnancement

Une classification formelle est nécessaire afin de distinguer les différents problèmes d'ordonnancement les uns des autres. Ce formalisme contient trois champs $\alpha|\beta|\gamma$ selon la notation proposée par Graham et al. (1979) et par Blazewicz et al. (1983) :

1. Le premier champ $\alpha$ représente l'organisation des ressources et est spécifié par la concaténation de deux éléments $\alpha_1$ et $\alpha_2$
   - $\alpha_1 \in \{\emptyset, P, Q, R, F, J, O\}$ représente la configuration de machines utilisées :
     - $\alpha_1 = \emptyset$ : problème à une seule machine.
     - $\alpha_1 \in \{P, Q, R\}$ : problèmes à machines parallèles.

- $\alpha_1$= P : les machines sont identiques.
- $\alpha_1$= Q : les machines sont uniformes.
- $\alpha_1$= R : les machines sont indépendantes.
- $\alpha_1$= F : atelier à cheminement unique (Flow shop).
- $\alpha_1$= J : atelier à cheminements multiples (Job shop).
- $\alpha_1$= O : atelier à cheminements libres : (Open shop).

– Le paramètre $\alpha_2 \in \{\emptyset, m\}$ permet de préciser le nombre de machines composant l'atelier.

- $\alpha_2$= $\emptyset$ : le nombre de machines est variable.
- $\alpha_2$= $m$ : le nombre de machines est constant et est égal à $m$.

2. Le champ $\beta$ précise les caractéristiques des jobs et des opérations et est généralement composé de sept sous-paramètres : $\beta_1$, $\beta_2$, $\beta_3$, $\beta_4$, $\beta_5$, $\beta_6$, $\beta_7$. Nous en citons quelques-uns.

– $\beta_1 \in \{\emptyset, \text{prmpt}\}$ renseigne sur l'autorisation ou non de la préemption :

- $\beta_1$= $\emptyset$ : la préemption n'est pas autorisée (lorsqu'une opération commence son exécution, elle doit être terminée sans interruption).
- $\beta_1$= prmpt : la préemption est autorisée (l'exécution d'une opération peut être interrompue).

– $\beta_2$ : caractérise les machines supplémentaires nécessaires à l'exécution d'un job (outils, ressources de transport).

- $\beta_2$= $\emptyset$ : indique qu'il n'y a pas de machines complémentaires.

– $\beta_3 \in \{\emptyset, \text{prec, tree, chain}\}$ décrit le type de précédence entre les jobs, i.e., le fait qu'un job doit être exécuté avant un autre. La valeur $\emptyset$ indique que les jobs sont indépendants. Les valeurs prec, tree, chain indiquent l'existence, respectivement, d'une relation de précédence générale, d'une relation de précédence sous forme d'arbre et d'une relation de précédence sous forme de chaîne.

– $\beta_4 \in \{\emptyset, r_i\}$ : spécifie si les jobs sont tous disponibles à l'état initial ou non.

– $\beta_5 \in \{\emptyset, p_i = \text{p}, p1 \leq p \leq p2\}$ détaille les durées des différentes opérations.

- $\beta_5 = \emptyset$ : les opérations ont des durées arbitraires.
- $\beta_5 = p_i = p$ : toutes les opérations ont des durées égales à $p$.
- $\beta_5 = p1 \leq p_i \leq p2$ : les durées des opérations sont comprises entre $p1$ et $p2$.

– $\beta_6 \in \{\emptyset, d_i, \widetilde{d}_i\}$ indique les éventuelles dates d'échéance (ou dates au plus tard) des jobs.

- $\beta_6 = \emptyset$ : les jobs n'ont pas de date d'échéance.

- $\beta_6 = d_i$ : chaque job à une date d'échéance de fin d'exécution sous peine de pénalisation.
- $\beta_6 = \widetilde{d}_i$ : chaque job à une date d'échéance impérative (date limite) qu'il faut absolument respecter.

– $\beta_7 \in \{\emptyset, \text{nwt}\}$ indique si un temps d'attente entre les différentes opérations d'un job est autorisé.

- $\beta_7 = \emptyset$ : un nombre illimité des opérations pouvant être en attente devant une machine.
- $\beta_7 = \text{nwt}$ (no-wait) toutes les opérations d'un job doivent être exécutées sans temps d'attente.

3. Le champ $\gamma$ correspond au critère à optimiser. (cf. chap1, §1.3.1.4).

## 1.4 Quelques notions de complexité

Nous rappelons dans cette section quelques définitions et notions de base sur la théorie de la complexité. Pour plus de détails, nous renvoyons le lecteur au très célèbre livre de Garey and Johnson (1979) qui propose une étude complète, et aussi à d'autre ouvrages sur la complexité (Barthélémy et al., 1992; Xuong, 1992; Cook, 1971; Karp, 1972).

La théorie de la complexité classe les problèmes combinatoires en plusieurs catégories. Parmi ces catégories, nous distinguons les problèmes d'optimisation et les problèmes de décision. Un problème de décision consiste à répondre à une question par un "OUI" ou par un "NON", alors qu'un problème d'optimisation impose de fournir une solution recherchée.

**Exemple 1.1.**
Un des premiers résultats établis en théorie de la complexité est le suivant :
***Problème de coloration graphe***
***Entrées*** : un graphe G et un entier $k$.
***Question*** : existe-t-il une $k$-coloration de G ?
***Complexité*** : NP-complet (Karp, 1972).

Chaque problème d'optimisation peut être décrit par un problème de décision. Il est alors possible d'analyser la difficulté d'un problème d'optimisation en considérant sa version en problème de décision.

Pour cela, la théorie de la complexité cherche à connaître la difficulté d'une réponse algorithmique à un problème. Autrement dit, peut-il y avoir un algorithme efficace pour répondre à la question de décidabilité pour un problème de décision ou de calculabilité pour un problème d'optimisation (Oulamara, 2009).

Dans le cas général, la résolution d'un problème se fait par un algorithme, tout en sachant

que ce dernier peut se traduire en un programme de machine de Turing [1] (Garey and Johnson, 1979). L'aspect qui nous intéresse dans la résolution d'un problème est sa complexité en temps, c'est-à-dire le nombre d'itérations que va mettre la machine pour résoudre le problème de décision.
A partir de ce nombre d'itérations, un algorithme est estimé efficace quand ce nombre est borné par un polynôme qui dépend de la taille des entrées, et il est dit dans ce cas polynomial. Par contre, si le nombre d'itérations est borné inférieurement par une fonction exponentielle de la taille des données, l'algorithme est dit alors exponentiel. Ceci motive l'introduction de la classe *P*.

*La classe P est l'ensemble des problèmes de décision solubles en temps polynomial par une machine de Turing déterministe. Autrement dit, les problèmes de décision pour lesquels il existe un algorithme polynomial permettant de répondre à la question.*

Nous rappelons que tout problème d'optimisation possède un problème de décision correspondant. Par exemple, le problème de décision associé au problème de coloration est présenté dans l'exemple 1.1. Cependant, nombreux problèmes de décision ne connaissent pas d'algorithmes polynomiaux permettant de les résoudre et le problème d'ordonnancement $J|p_{ij} = 1|C_{max}$ en est un exemple. Ceci motive la définition de la classe *NP*.

On peut caractériser les problèmes de cette classe (*NP*) comme des problèmes solubles en temps polynomial par une machine de Turing d'un type plus général, appelée plus précisément machine de Turing non déterministe.

*Un problème de décision appartient à la classe NP s'il est soluble par une machine de Turing non déterministe polynomiale. Autrement dit, les problèmes de décision pour lesquels il existe un algorithme tel que si on lui fournit une solution dont la réponse est "OUI", il est capable de le vérifier en temps polynomial.*

Partant de l'idée de l'inclusion de *P* dans *NP* étant évidente et de la difficulté de répondre à la question $P = NP$?, la théorie de complexité s'intensifie particulièrement à étudier les relations entre problèmes dans le sens de leur difficulté afin d'établir une notion de hiérarchie de difficulté relative à la résolution des problèmes.
C'est dans ce contexte que la notion de réduction de Karp a été introduite, où un problème de décision $D_1$ est dit réductible à un autre problème de décision $D_2$ s'il existe une fonction polynomiale $f$ qui transforme chaque énoncé de $D_1$ en un autre énoncé de $D_2$ de telle manière que : la réponse pour $D_1$ est oui si et seulement si la réponse pour $D_2$ est oui.
Autrement dit, une réduction de Karp d'un problème $D_1$ à un problème $D_2$ indique que le problème $D_2$ n'est pas plus facile à résoudre que le problème $D_1$, en ce sens que si l'on savait résoudre $D_2$ en temps polynomial, on saurait également résoudre $D_1$ en temps polynomial.

Toutefois, il semble moins probable qu'un problème donné soit polynomial s'il y a d'autres

1. cf. Garey and Johnson (1979) pour la définition d'une machine de Turing.

problèmes (sans que leur statut d'être polynomiaux ou non soit déterminé) qui se réduisent à ce problème. Ceci explique la motivation de savoir si ce problème n'est pas plus facile que tous les autres problèmes *NP* (i.e. si tout problème de *NP* se réduit à lui). Ce problème est appelé complet pour la classe *NP*, et est défini comme suit :

*Un problème de décision est dit NP-complet si tout problème de NP se réduit à lui.*

De plus, un problème d'optimisation est dit NP-difficile si le problème de décision qui lui est associé est NP-complet. Dans ce contexte, une longue liste de problèmes NP-complets et NP-difficiles est disponible dans le livre de Garey and Johnson (1979).

De cette étude, il en résulte que pour résoudre un problème d'optimisation combinatoire $\Pi$, il faut tout d'abord analyser le problème de décision équivalent. Si ce dernier appartient à la classe P, alors $\Pi$ est polynomial. Par contre, si le problème de décision est NP-Complet, alors le problème $\Pi$ est NP-difficile. Dans ce cas, la recherche d'une solution optimale peut occasionner une explosion combinatoire, et conduit à une croissance exponentielle du temps de recherche. Cependant, pour des instances de problèmes de petite taille, une méthode exacte peut être utilisée. Dans le cas contraire, le recours à des méthodes de résolution approchées s'impose, comme les heuristiques, les métaheuristiques, etc.

## 1.5 Conclusion

Ce chapitre a été essentiellement consacré au rappel de quelques notions de bases relatives aux problèmes de coloration de graphes et aux problèmes d'ordonnancement. Nous avons mis en évidence l'intérêt de la coloration des graphes mixtes qui permet de modéliser les problèmes d'ordonnancement avec des contraintes de précédence. Ensuite, quelques définitions de la théorie de complexité sont présentées pour justifier la nécessité de l'étude préalable de la complexité des problèmes d'optimisation combinatoire avant leur résolution. Dans le chapitre qui suit, nous présentons les deux problématiques d'ordonnancement traitées dans le cadre de cette thèse, et nous décrivons leur réduction à des problèmes de coloration de graphes mixtes.

# Chapitre 2

# Problématique et état de l'art

## Sommaire

## 2.1 Introduction

Dans ce chapitre, nous considérons deux types de problèmes d'ordonnancement job shop avec des opérations de durée unitaire, nous décrivons leur équivalence aux problèmes de coloration de graphes mixtes. Puis, nous présentons une étude de la complexité des deux problèmes traités, en plus de ceux qui leur sont particuliers ou leur sont plus généraux. Ensuite, un état de l'art résumant les principaux travaux de la littérature en relation avec les problèmes étudiés est donné.

## 2.2 Problématique et notation

Le problème d'ordonnancement job shop avec des opérations de durée unitaire est décrit par la donnée d'un ensemble J de $n$ jobs à réaliser sur un ensemble M de $m$ machines. Chaque job $J_i$ ($i = 1, ..., n$) est composé d'un ensemble ordonné de $n_i$ opérations $J_i = \{O_{i1}, O_{i2}, ..., O_{in_i}\}$ pour lesquelles des contraintes de précédence sont définies selon les gammes opératoires. A chaque opération $O_{ik}$ ($i = 1, ..., n; k = 1, ..., n_i$) est associée une durée de traitement unitaire $p_{ik} = 1$, et ne peut être exécutée que par une seule machine. Chaque machine $M_j$, ($j = 1, ..., m$) ne peut traiter qu'au plus une opération à la fois.

Pour ce problème, deux hypothèses sont considérées :

1. Un job peut visiter une même machine plusieurs fois (réentrant).
2. Chaque job visite chaque machine au plus une fois (non-réentrant).

L'objectif est de déterminer l'ordre de passage des opérations sur les différentes machines, tout en respectant les contraintes du problème et en optimisant un critère donné.

Dans cette étude, nous nous intéressons à optimiser deux objectifs distincts :

1. le makespan $C_{max} = \max_{i \in 1,...,n} C_{in_i}$.
2. la somme des dates de fin des jobs $\sum C_i = \sum_{i \in \{1,...,n\}} C_{in_i}$.

où $C_{in_i}$ dénote la date de fin d'exécution de la dernière opération $O_{in_i}$ du job $J_i$.

De ces deux critères, deux problèmes d'ordonnancement différents sont donc considérés, à savoir $J|p_{ij} = 1|C_{max}$ et $J|p_{ij} = 1|\sum C_i$.

### 2.2.1 Formulation

Formellement, considérons l'ensemble O constitué de toutes les opérations du problème d'ordonnancement et $k$ un entier positif, le problème d'ordonnancement job shop avec des durées unitaires peut être défini comme une application $\Phi : O \mapsto \{1, ..., k\}$ tel que, pour chaque paire d'opérations distinctes $O_{ij}$ et $O_{hl}$ de l'ensemble O, les deux conditions suivantes sont vérifiées :

1. si $(O_{ij}, O_{hl})$ appartenant au même job $J_i$ (i.e. $i = h$) et $O_{ij}$ précède $O_{hl}$ (i.e. $j < l$), alors $\Phi(O_{ij}) < \Phi(O_{hl})$.
2. si $(O_{ij}, O_{hl})$ s'exécutent sur une même machine, alors $\Phi(O_{ij}) \neq \Phi(O_{hl})$.

Notre but est de trouver une application $\Phi$, qui satisfait les deux types de contraintes suscitées.
Il en découle les deux critères que nous étudions de manière distincte et qui sont :

1. minimiser $\max_{i \in \{1,\dots,n\}} \Phi(O_{in_i})$.
2. minimiser $\sum_{i \in \{1,\dots,n\}} \Phi(O_{in_i})$.

Partant de cette formulation, nous pouvons constater l'équivalence entre le problème d'ordonnancement job shop avec des opérations de durée unitaire et le problème de coloration de graphes mixtes. Les descriptions peuvent aussi être retrouvées dans les travaux de Sotskov et al. (2001), Al-Anzi et al. (2006) et Kouider et al. (2017).

Plus précisément, considérons un graphe mixte $G = (V, E, A)$, où les ensembles non vides V, E et A représentent respectivement les sommets, les arêtes et les arcs. Ce graphe mixte est obtenu par la représentation de ce qui suit :

- une opération $O_{ik}$ par un sommet $v_{ik}$
- une relation de précédence directe entre deux opérations $O_{ik}$ et $O_{i(k+1)}$ d'un même job $J_i$ par un arc $(v_{ik}, v_{i(k+1)})$ avec une extrémité initiale $v_{ik}$ et une extrémité terminale $v_{i(k+1)}$
- une contrainte de disjonction entre deux opérations $O_{ik}$ et $O_{hl}$ d'une même machine par une arête $[v_{ik}, v_{hl}]$ joignant deux sommets $O_{ik}$ et $O_{hl}$
- les opérations d'un job $J_i$ par des sommets d'un chemin de longueur maximale, noté $G_i = (V_i, \emptyset, A_i)$, où $V_i \subset V$ et $A_i \subset A$
- les opérations assignées à une même machine $M_j$ par des sommets d'une clique maximale, notée $G^j = (V^j, E^j, \emptyset)$, où $V^j \subset V$ et $E^j \subset A$
- la date de fin d'exécution d'une opération $O_{ik}$ par une couleur assignée au sommet $v_{ik}$
- la minimisation du makespan par le nombre chromatique (noté, $\chi$)
- la minimisation de la somme des fins d'exécution des jobs par la minimisation de la somme des couleurs des extrémités terminales des chemins de longueur maximale (qu'on note $\sigma$)

A l'issu des correspondances que nous avons défini et compte tenu qu'une opération ne peut appartenir qu'à un seul job et ne peut être exécutée que par une seule machine, il en résulte que tout sommet du graphe mixte ne peut appartenir qu'à un seul chemin de longueur maximale et qu'à une seule clique maximale.

Par conséquent, le graphe partiel engendré que par l'ensemble des arcs ($G = (V, \emptyset, A)$) est une union disjointe de chemins de longueur maximale, et le graphe partiel engendré que par l'ensemble d'arêtes ($G = (V, E, \emptyset)$) est une union disjointe de cliques maximales (i.e. $G = (V, \emptyset, A) = \bigcup_{i=1}^{i=n}(G_i = (V_i, \emptyset, A_i))$ et $G = (V, E, \emptyset) = \bigcup_{j=1}^{j=m}(G^j = (V^j, E^j, \emptyset))$.

Pour la résolution du problème de coloration de graphes mixtes avec le critère $\chi$ (resp. $\sigma$) modélisant l'ordonnancement $J|p_{ij} = 1|C_{max}$ (resp. $J|p_{ij} = 1|\sum C_i$), deux hypothèses sont considérées :

- H1 : Un chemin de longueur maximale peut visiter une même clique maximale plusieurs fois.
- H2 : Chaque chemin de longueur maximale visite chaque clique de cardinalité maximale au plus une fois.

**Exemple 2.1.**
Considérons les données du problème d'ordonnancement $\pi_{2.1}$ présentées dans le tableau 2.1. Ce problème comporte 3 jobs $\{J_1, J_2, J_3\}$ à ordonnancer sur 3 machines $\{M_1, M_2, M_3\}$. Chaque job est constitué de 3 opérations.

TABLEAU 2.1 – Les données du problème d'ordonnancement $\pi_{2.1}$

| Job | $1^{er}$ passage | | $2^{ième}$ passage | | $3^{ième}$ passage | |
|---|---|---|---|---|---|---|
| | Machine | Opération | Machine | Opération | Machine | Opération |
| $J_1$ | $M_1$ | $O_{11}$ | $M_3$ | $O_{12}$ | $M_2$ | $O_{13}$ |
| $J_2$ | $M_1$ | $O_{21}$ | $M_2$ | $O_{22}$ | $M_3$ | $O_{23}$ |
| $J_3$ | $M_1$ | $O_{31}$ | $M_2$ | $O_{32}$ | $M_3$ | $O_{33}$ |

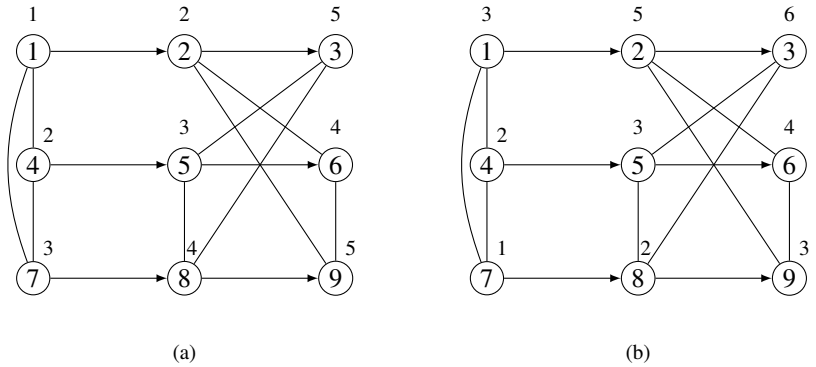


FIGURE 2.1 – Colorations du graphe mixte associé au problème d'ordonnancement $\pi_{2.1}$

La figure 2.1 présente deux colorations pour le graphe mixte associé au problème d'ordonnancement. Ce graphe mixte est composé de 9 sommets, chacun représentant une opération du problème, de 3 chemins de longueur maximale représentant les opérations des jobs à exécuter dans un ordre bien déterminé, et enfin de 3 cliques maximales représentant les opérations à exécuter par une même machine.

Dans les deux colorations proposées pour le problème $\pi_{2.1}$, si nous focalisons sur le critère $C_{max}$ alors la coloration du graphe mixte donnée par la sous-figure ( 2.1.a) est meilleure que celle présentée dans la sous-figure ( 2.1.b). Par contre si le critère $\sum C_i$ est considéré, nous pouvons constater que la coloration du graphe mixte donnée par la sous-figure ( 2.1.b) est meilleure que celle présentée dans la sous-figure ( 2.1.a).

### 2.2.2 Motivation

L'ordonnancement job shop avec des opérations de durée unitaire trouve applications dans plusieurs domaines pratiques. Nous citons comme exemple, l'ordonnancement des examens au niveau des universités (quand les examens écrits passent avant les épreuves orales), l'ordonnancement des compétitions sportives quand les jeux possèdent la même durée, et enfin l'ordonnancement des procédures médicales au niveau des hôpitaux (Sotskov et al., 2001).

Nous pouvons aussi rencontrer ce type de problèmes dans certains domaines industriels très pointus. Nous citons à titre d'exemple en se référant aux travaux de Guschinskaya (2007), l'exemple étudié sur la conception en avant-projet des systèmes d'usinage dédiés à la production en grande série. L'auteur s'est intéressé aux problèmes d'optimisation de la configuration de trois types de systèmes d'usinage, à savoir les machines de transfert, les machines à table mobile et les machines à table circulaire pivotante.

La machine de transfert, selon l'auteur, est constituée d'une séquence de postes de travail reliés par un dispositif de transfert de pièces. Ainsi, chaque pièce chargée sur la machine passe par tous les postes de travail installés, selon leur disposition. Une pièce brute est chargée sur le premier poste, puis elle traverse la machine et subit des opérations d'usinage sur chaque poste de travail et est transformée en une pièce finie qui sera déchargée du dernier poste de travail. A intervalle de temps régulier, égal au temps de cycle, toutes les pièces se trouvant sur la machine sont simultanément déplacées vers le poste de travail suivant. Au même moment, une nouvelle pièce brute est chargée sur le premier poste de travail alors qu'une pièce finie est déchargée du dernier poste.

La configuration de telles machines est déterminée à la fois par le nombre de postes de travail, la composition de chaque poste en termes de boîtiers (leur nombre et l'ordre d'activation), et par les blocs d'opérations affectés à chaque boîtier. Le problème d'optimisation de la configuration de telles machines consiste alors à répartir les opérations d'usinage à des boîtiers d'usinage et à des postes de travail de manière à ce que toutes les contraintes techniques et technologiques soient respectées, et que le coût de la machine soit le plus petit possible. Ainsi pour déterminer le nombre exact de blocs nécessaires afin d'affecter tous les prédécesseurs directs d'une opération en respectant les contraintes de précédence et d'exclusion, il faut résoudre un problème similaire au problème d'ordonnancement job shop avec des opérations de durée unitaire optimisant le makespan (Guschinskaya, 2007).

## 2.3 Etudes de complexité

Pour commencer, rappelons que lorsque les machines sont visitées au plus une fois par chaque job, le problème d'ordonnancement optimisant $C_{max}$ (resp. $\sum C_i$) est noté $J|p_{ij} = 1|C_{max}$ (resp. $J|p_{ij} = 1|\sum C_i$). Cependant lorsque les machines peuvent être visitées plus d'une fois par chaque job, la notation $J|p_{ij} = 1, rep|C_{max}$ (resp. $J|p_{ij} = 1, rep|\sum C_i$) est utilisée.

Il est évident de constater que le problème $J|p_{ij} = 1, rep|C_{max}$ (resp. $J|p_{ij} = 1, rep|\sum C_i$) est une généralité du problème $J|p_{ij} = 1|C_{max}$ (resp. $J|p_{ij} = 1|\sum C_i$).

De plus dans le cas où chaque job ne peut revisiter la même machine que par des opérations successives, le problème $J|p_{ij} = 1, rep|C_{max}$ (resp. $J|p_{ij} = 1, rep|\sum C_i$) devient équivalent au job shop préemptif $J|prmpt|C_{max}$ (resp. $J|prmpt|\sum C_i$) (i.e. $p_{ij} \geq 1$)(Sotskov et al., 2001).

Nous partons de l'équivalence entre ces deux types de problèmes pour étudier la complexité des problèmes $J|p_{ij} = 1, rep|C_{max}$ et $J|p_{ij} = 1, rep|\sum C_i$. La complexité des problèmes d'ordonnancement préemptif est largement étudiée dans la littérature. Sotskov dans (Sotskov, 1985, 1991) a proposé un algorithme de complexité $O(r^3)$ pour les problèmes $J|n = 2, prmpt|C_{max}$ et $J|n = 2, prmpt|\sum C_i$, où $r = \max_{i \in \{1,\ldots,n\}} n_i$. L'algorithme de Jackson (Jackson, 1956) peut également être utilisé pour résoudre le problème $J2|n_i \leq 2, prmpt|C_{max}$ parce que les préemptions ne réduisent pas la valeur optimale du $C_{max}$ (Brucker, 1999).

D'autre part, Gonzales and Sahni (1978) ont prouvé que les problèmes $J2|n_i \leq 3, prmpt|C_{max}$ et $J2|prmpt|C_{max}$ sont NP-difficiles. Sotskov and Shakhlevich (1995) ont montré que $J3|n = 3, prmpt|C_{max}$ et $J3|n = 3, prmpt|\sum C_i$ sont NP-difficiles.
Brucker (1999) a amélioré les résultats de complexité en démontrant que $J2|n = 3, prmpt|C_{max}$ et $J2|n = 3, prmpt|\sum C_i$ sont NP-difficiles.

Considérons maintenant le cas où le passage répété des jobs sur une même machine n'est pas autorisé. Le problème $J|p_{ij} = 1|C_{max}$ a été montré, par Lenstra and Rinnooy Kan (1979), NP-difficile pour $m > 2$, et est également NP-difficile pour le cas particulier $J3|p_{ij} = 1|C_{max}$. Sinon pour le cas particulier de deux machines ($J2|p_{ij} = 1|C_{max}$), Hefetz and Adiri (1982) ont proposé un algorithme de complexité linéaire par rapport au nombre d'opérations pour sa résolution.

Pour ce qui est du problème $J|p_{ij} = 1|\sum C_i$, notons que ce dernier est une généralité du problème $F|p_i\{0,1\}|\sum C_i$. La notation {0,1} signifie que la durée de l'opération peut être égale à zéro ou à une unité. Le problème $F|p_i\{0,1\}|\sum C_i$ est NP-difficile (Gonzalez, 1982), il en découle que le problème $J|p_{ij} = 1|\sum C_i$ est aussi NP-difficile dans la mesure où le problème $F|p_i\{0,1\}|\sum C_i$ est un cas particulier du problème $J|p_{ij} = 1|\sum C_i$.

## 2.4 État de l'art

Le but de cette section est de présenter quelques travaux qui se rapprochent de la problématique étudiée dans ce manuscrit. Nous commençons par présenter quelques résultats sur la coloration de graphes mixtes, puis d'autres résultats sur des problèmes d'ordonnancement avec des opérations de durée unitaire en général. A la fin, nous décrivons les travaux sur l'ordonnancement job shop avec des opérations de durée unitaire où les différentes bornes inférieures et supérieures proposées pour les critères à optimiser sont détaillées.

### 2.4.1 Coloration de graphes mixtes

Rappelons que le problème de minimisation du nombre de couleurs dans la coloration d'un graphe non orienté donné (nombre chromatique) est le type de problèmes le plus utilisé de la coloration des sommets, et est montré NP-difficile (Garey and Johnson, 1979). Cependant, ce type de coloration connaît des limites et ne permet pas de prendre en compte les diverses exigences imposées par les problèmes réels d'ordonnancement. Il est donc nécessaire d'étendre ce modèle de colorations à d'autres exigences (Hansen et al., 1997). Comme nous l'avons déjà signalé et à titre d'exemple, la coloration des sommets d'un graphe non orienté ne permet pas de modéliser certains problèmes d'ordonnancement ayant à la fois des conditions d'incompatibilité et des contraintes de précédence. Considérant ce type de problèmes d'ordonnancement, Sotskov et al. (1976) ont introduit un nouveau modèle plus général capable de prendre en compte ces exigences, à savoir la coloration des graphes mixtes.

De toute évidence, la coloration des sommets d'un graphe mixte est plus générale que celle des sommets d'un graphe non orienté, et est par conséquent NP-difficile. Parmi les travaux de la littérature qui ont étudié ce type de problèmes, nous citons les travaux de Hansen et al. (1997) qui ont proposé un algorithme de complexité $O(n^2)$ pour la coloration des arbres mixtes. Par la suite, et considérant le même problème, Furmanczyk et al. (2008) ont proposé un nouveau algorithme polynomial pour le calcul du nombre chromatique des arbres mixtes, et leur algorithme a amélioré celui proposé par Hansen et al. (1997).

Ries (2007) dans son travail s'est intéressé à la complexité du problème de coloration de certaines classes spéciales de graphes mixtes. Il a montré que ce problème est NP-complet pour les graphes bipartis planaires et pour les graphes bipartis ayant un degré maximum 3.

Ries and de Werra (2008) ont étudié deux types différents de coloration de graphes mixtes. Le premier impose aux sommets adjacents d'avoir des couleurs différentes, et au sommet de l'extrémité initiale d'un arc d'avoir une couleur inférieure à celle du sommet de l'extrémité terminale. Dans le second type de coloration, les sommets liés par un arc sont contraints d'avoir la même couleur. Pour ces deux types de coloration, les auteurs ont proposé des bornes pour le nombre chromatique.

### 2.4.2 Ordonnancement à durées unitaires

Les problèmes d'ordonnancement à durées unitaires sont largement étudiés dans la littérature. Nous citons, les travaux de Rebaine (2010) qui a proposé deux heuristiques pour le problème d'ordonnancement de type flow shop avec des temps de préparation, et des opérations de durée unitaire. Pour le cas réduit à deux machines de ce problème, Moukrim et al. (2014) ont proposé un algorithme branch and bound utilisant de nouvelles bornes inférieures et supérieures pour le critère $C_{max}$.
Gerstl and Mosheiov (2014) ont étudié le problème d'ordonnancement flow shop flexible à deux étages avec des opérations de durée unitaire. Pour sa résolution, les auteurs ont proposé des variantes d'algorithmes exacts basés sur la programmation dynamique, et ont considéré deux objectifs à minimiser ($\sum C_i$ et $C_{max}$).

Timkovsky (1998) a réalisé une étude comparative entre les problèmes d'ordonnancement job shop à durées unitaires et les problèmes d'ordonnancement à machines parallèles. Dans son papier intitulé *"Is a unit-time job shop not easier than identical parallel machines ?"*, il a fourni une réponse positive à la question posée pour une large classe de critères de minimisation incluant : le makespan, le retard maximum, la somme des dates de fin des jobs, la somme pondérée des dates de fin des jobs, la somme des retards, la somme pondérée des retards, le nombre de jobs en retard et le nombre pondéré des jobs en retard.

### 2.4.3 Ordonnancement job shop à durées unitaires par la coloration de graphes mixtes

Pour le problème d'ordonnancement job shop avec des opérations de durée unitaire, Kouider et al. (2014) et Kouider (2014) ont étudié le problème $J|p_{ij} = 1|C_{max}$ par la coloration de graphes mixtes et ont proposé une formulation mathématique sous forme de deux programmes linéaires à variables mixtes. Ces programmes sont basés sur des contraintes disjonctives qui engendrent une explosion du nombre de cas à traiter séparément lorsque le nombre de disjonctions augmente. Les auteurs ont effectué des tests expérimentaux sur des benchmarks modifiés de moins de 100 sommets. Pour une résolution approchée du problème, ils ont proposé quatre variantes de l'algorithme du recuit simulé. Cependant, l'efficacité de ces variantes en terme de qualité de solution est estimée moins bonne comparativement à l'efficacité de l'algorithme tabu search proposé dans (Kouider et al., 2015) ou dans sa version améliorée proposée dans (Kouider et al., 2017).

Pour le même problème, Sotskov et al. (2001) ont proposé un algorithme branch and bound pour la coloration de graphes mixtes modélisant le problème $J|p_{ij} = 1|C_{max}$. Les solutions du problème sont codées dans une structure de données de type arbre. Chaque noeud de cet arbre, qui représente une solution partielle du problème, contient un unique vecteur W de $n$ composantes. La $i$è$me$ composante de W représente la couleur assignée au job $j_i$. Pour trai-

ter le conflit d'affectation d'une couleur "$c$" à des sommets appartenant à des jobs différents, l'algorithme attribue la couleur "$c$" à la composante qui correspond à un de ces jobs et "$-c$" aux composantes des autres jobs.
La même démarche est adoptée par Al-Anzi et al. (2006) pour la coloration de graphes mixtes modélisant le problème $J|p_{ij}=1|\sum C_i$.

Plusieurs bornes inférieures et supérieures ont été proposées dans ces deux travaux de référence. Dans la suite, nous présentons en détail ces bornes, et nous retenons les meilleures d'entre elles dans le but de les utiliser dans des variantes d'algorithmes que nous proposons dans ce présent travail, et aussi pour les comparer avec les bornes que nous proposons dans les chapitres qui suivent.

*Borne supérieure et bornes Inférieures proposées par Sotskov et al. (2001)*

Une borne supérieure et deux bornes inférieures ont été proposées par les auteurs pour le nombre chromatique $\chi$.
La borne supérieure $UB_{\chi 0}$ est donnée par la formule 2.1

$$UB_{\chi 0} = \sum_{i=1}^{i=n} |V_i| \tag{2.1}$$

Une première borne inférieure $LB_{\chi 1}$, donnée par la formule 2.2, est globale et nécessite la récupération des données à partir des autres noeuds de l'arbre de solutions.

$$LB_{\chi 1} = \max_{M_k \in M} \{\min_{j_d \in J} (h_k^d) + |V^k| + \min_{j_d \in J} (t_k^d)\} \tag{2.2}$$

où, $h_k^d$ (resp. $t_k^d$) est le nombre d'opérations précédant (resp. succédant) une opération d'un job $J_d$ affectée à la machine $M_k$.

Une seconde borne inférieure $LB_{\chi 2}$, donnée par la formule 2.3, est locale. Son calcul ne nécessite que les données locales disponibles au niveau du vecteur W associé à un noeud de l'arbre, ainsi que le nombre d'opérations $n_i$ de chaque job $J_i$. Il a été noté que malgré la rapidité de calcul de $LB_{\chi 2}$, cette borne demeure dans la plupart des cas moins efficace que $LB_{\chi 1}$.

$$LB_{\chi 2} = \max_{j_d \in J} (n_i + l_k) \tag{2.3}$$

où $l_k$ est le nombre de couleurs qui ont été omises par certains sommets du chemin ($G_k = (V_k, \emptyset, A_k)$) durant le processus de coloration au niveau d'un noeud de l'arbre de solutions.

*Borne supérieure et bornes Inférieures proposées par Al-Anzi et al. (2006)*

Pour le critère $\sigma$, les auteurs ont proposé une borne supérieure $UB_{\sigma 0}$ et trois bornes infé-

rieures ($LB_{\sigma 0}, LB_{\sigma 1}, LB_{\sigma 2}$). La borne supérieure $UB_{\chi 0}$ est donnée par la formule 2.4.

$$UB_{\sigma 0} = \sum_{i=1}^{i=n} (n+1-i)\, n_i \tag{2.4}$$

La borne inférieure $LB_{\sigma 0}$ est locale, et les deux bornes inférieures $LB_{\sigma 1}$ et $LB_{\sigma 2}$ nécessitent des données globales, c'est-à-dire des données situées en dehors du noeud courant de l'arbre de solutions. Ces trois bornes sont détaillées comme suit :

*Borne inférieure* $LB_{\sigma 0}$

$$LB_{\sigma 0} = \sum_{i=1}^{i=n} n_i + l_i^j \tag{2.5}$$

où $l_i^j$ est le nombre de couleurs qui ont été omises par certains sommets du chemin ($G_i = (V_i, \emptyset, A_i)$) durant le processus de coloration au niveau du noeud de l'arbre contenant le vecteur de coloration $w^j$.

Il est à noter que seules les données locales sont nécessaires pour calculer $LB_{\sigma 0}$. En effet, il suffit seulement de connaître le vecteur de coloration $w^j$ déterminant l'état actuel du processus de coloration, et le vecteur de données $(n_1, n_2, ..., n_n)$. Ceci explique la rapidité du calcul de cette borne inférieure locale $LB_{\sigma 0}$, contrairement aux bornes inférieures globales $LB_{\sigma 1}$ et $LB_{\sigma 2}$ qui nécessitent beaucoup plus de temps de calcul du fait qu'elles nécessitent la récupération des données à partir des différents noeuds de l'arbre de solutions.

La borne $LB_{\sigma 1}$ est donnée par la formule 2.6 :

$$LB_{\sigma 1} = \max_{M_k \in M} \{ n \min_{j_d \in J} (h_k^d) + \frac{n(n+1)}{2} + n \min_{j_d \in J} (t_k^d) \} \tag{2.6}$$

où, $h_k^d$ (resp. $t_k^d$) est le nombre d'opérations précédant (resp. succédant) une opération d'un job $J_d$ affectée à la machine $M_k$.
Cette borne n'est valide que dans le cas où chaque machine traite chaque job exactement une fois.

Selon les hypothèses du problème, cette condition n'est pas toujours satisfaite puisqu'il est possible qu'au moins un job ne visite pas une des machines, ou bien qu'il visite plus d'une fois la même machine. Dans ce cas particulier, la borne inférieure $LB_{\sigma 1}$ a été adaptée par les auteurs pour obtenir la formule 2.7 :

$$LB_{\sigma 1} = \max_{M_k \in M} \{ s \min_{j_d \in J} (h_k^d) + \sum_{J_i \in J(M_k)} C_i^* + s \min_{j_d \in J} (t_k^d) \} + \sum_{J_i \in J \setminus J(M_k)} n_i \} \tag{2.7}$$

Pour calculer cette borne, deux types d'ensembles de jobs sont distingués pour chaque machine $M_k$. Le premier $J(M_k)$ regroupe les $s$ jobs qui passent par la machine $M_k$ et le second ensemble regroupe les autres jobs, i.e. les jobs de l'ensemble $J \setminus J(M_k)$.

Dans la formule 2.7, la valeur $\sum_{J_i \in J(M_k)} C_i^*$ est obtenue en ordonnançant les opérations affectées à la machine $M_k$ par ordre non croissant des nombres d'opérations par job.

La troisième borne inférieure globale $LB_{\sigma 2}$ est une généralisation de la borne inférieure locale $LB_{\sigma 0}$. Soit $n(c)$ le plus grand nombre possible de jobs dont les opérations peuvent être colorées par la couleur $c$ lors du déroulement de l'algorithme, et considérons $m(c)$ le nombre de machines nécessaires pour traiter les opérations dans l'intervalle de temps [c-1, c]. Ainsi, la borne inférieure $LB_{\sigma 2}$ est donnée par la formule 2.8 :

$$LB_{\sigma 2} = \sum_{i=1}^{n} n_i + \sum_{j=1}^{r} \max\{0, n(j) - m(j)\} \tag{2.8}$$

où $r = max\{n_i : J_i \in J\}$.

En se basant sur ces trois bornes inférieures, les auteurs ont développé trois variantes de l'algorithme branch and bound. Ils ont constaté, d'après les résultats des tests expérimentaux, que la variante basée sur $LB_{\sigma 1}$ est meilleure que les deux autres variantes basées sur les deux autres bornes.

De plus, leurs tests expérimentaux ont été réalisés sur des instances générées aléatoirement en vérifiant les deux hypothèses du problème. En d'autres termes, lorsque les machines sont visitées au plus une fois par chaque job, la borne $LB_{\sigma 1}$ donnée par la formule 2.6 est utilisée, alors que s'il existe des machines qui sont visitées plus d'une fois par des jobs, la borne $LB_{\sigma 1}$ donnée par la formule 2.7 est utilisée.

Cependant, nous avons constaté que $LB_{\sigma 1}$ de la formule 2.7 ne peut être toujours vraie comme nous le montrons à travers le contre exemple 2.2 suivant.

**Exemple 2.2.**
Considérons le problème d'ordonnancement $\pi_{2.2}$ composé de 2 jobs $\{J_1, J_2\}$ à ordonnancer sur 2 machines $\{M_1, M_2\}$. Le premier job est constitué de 5 opérations, et le second est constitué de 3 opérations. Leur ordre de passage sur les deux machines est détaillé dans le tableau 2.2 :

TABLEAU 2.2 – Les données du problème d'ordonnancement $\pi_{2.2}$

| Job | $1^{er}$ passage | | $2^{ième}$ passage | | $3^{ième}$ passage | | $4^{ième}$ passage | | $5^{ième}$ passage | |
|---|---|---|---|---|---|---|---|---|---|---|
| | M | O | M | O | M | O | M | O | M | O |
| $J_1$ | $M_2$ | $O_{11}$ | $M_1$ | $O_{12}$ | $M_1$ | $O_{13}$ | $M_1$ | $O_{14}$ | $M_2$ | $O_{15}$ |
| $J_2$ | $M_2$ | $O_{21}$ | $M_1$ | $O_{22}$ | $M_2$ | $O_{23}$ | | | | |

La figure 2.2 présente le graphe mixte associé à ce problème.

Procédons maintenant au calcul de la borne $LB_{\sigma 1}$ de la formule 2.7.

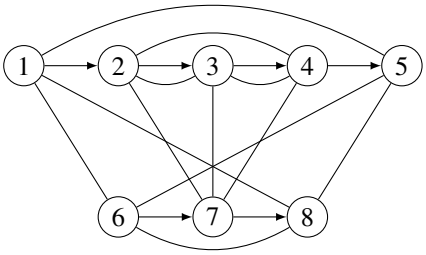


FIGURE 2.2 – Graphe mixte associé au problème d'ordonnancement $\pi_{2.2}$

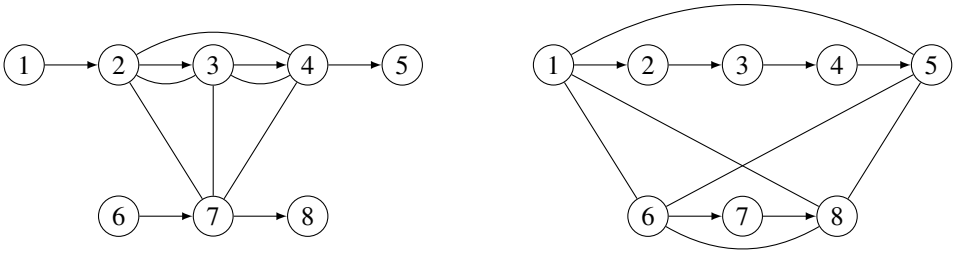


FIGURE 2.3 – Graphes partiels engendrés par les cliques maximales

Pour illustrer le calcul, nous considérons les deux graphes partiels présentés dans la figure 2.3. Le premier (resp. le second) graphe partiel modélise les données associées à la première (resp. seconde) machine et est engendré par tous les arcs du graphe mixte et par les arêtes de la première (resp. seconde) clique maximale.

Le calcul de la valeur de $LB_{\sigma 1}$ associé au graphe mixte de la figure 2.2 est comme suit :

pour $k = 1$ on a (cf. figure 2.2.a) :

$$\min_{j_d \in J}(h_1^d) + \sum_{J_i \in J(M_1)} C_i^* + \min_{j_d \in J}(t_1^d) + \sum_{J_i \in J \setminus J(M_1)} n_i = 2\min(1,1) + (3+4) + 2\min(1,1) + 0 = \mathbf{11}.$$

pour $k = 2$ on a (cf. figure 2.2.b) :

$$\min_{j_d \in J}(h_2^d) + \sum_{J_i \in J(M_2)} C_i^* + \min_{j_d \in J}(t_2^d) + \sum_{J_i \in J \setminus J(M_2)} n_i = 2\min(0,0) + (2+4) + 2\min(0,0) + 0 = \mathbf{6}.$$

Ainsi, $LB_{\sigma 1} = \max(11,6) = \mathbf{11}$. Cependant, le graphe mixte associé au problème peut être coloré par une coloration de coût égal à 9 (cf. figure 2.4).

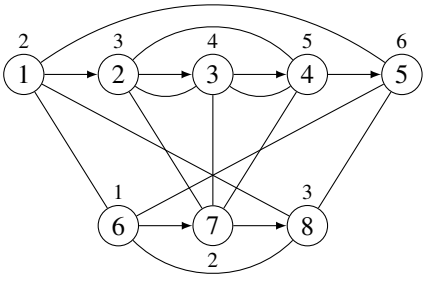


FIGURE 2.4 – Coloration du graphe mixte associé au problème d'ordonnancement $\pi_{2.2}$

## 2.5 Conclusion

Nous avons dans ce chapitre présenté une description des deux problèmes d'ordonnancement job shop avec des durées unitaires qui se distinguent par le critère à optimiser. Le premier optimise le makespan et le second la somme des dates de fin d'exécution des jobs. Ensuite, nous les avons formalisé par la coloration d'une classe spéciale de graphes mixtes qui se caractérise par deux types de graphes partiels. Le premier graphe partiel engendré par l'ensemble des arcs constitue une union disjointe de cliques maximales, et le second graphe partiel engendré par les arêtes constitue une union disjointe de chemins de longueur maximale. Ainsi, le problème d'ordonnancement avec des durées unitaires minimisant le makespan (resp. $\sum C_i$) est équivalent au problème de coloration de graphes mixtes avec le critère $\chi$ (resp. $\sigma$).

D'une manière générale, les problèmes d'ordonnancement job shop sont largement étudiés et sont connus pour leur NP-difficulté, même pour leur cas particulier où la durée d'exécution de chaque opération est unitaire. De ce fait, une étude de la complexité et un état de l'art de quelques travaux sur les problèmes d'ordonnancement job shop à durées unitaires sont présentés.

Dans le chapitre suivant, nous décrivons les bornes supérieures et inférieures ainsi que les méthodes de résolution que nous proposons pour la résolution des deux problèmes de coloration de graphes mixtes.

# Chapitre 3

# Méthodes de résolution

## Sommaire

## 3.1 Introduction

Dans ce chapitre, nous nous intéressons à la résolution des deux problèmes de coloration de graphes mixtes modélisant les problèmes d'ordonnancement job shop à durées unitaires décrits dans le chapitre précédent. Le premier problème de coloration s'intéresse au nombre chromatique ($\chi$) et le second au critère de minimisation de la somme des couleurs des extrémités terminales des chemins de longueur maximale ($\sigma$). Nous présentons dans un premier temps la borne inférieure et la borne supérieure proposées pour le critère ($\chi$), et que nous adaptons pour le critère ($\sigma$). Nous décrivons ensuite, dans la section 4, les modèles mathématiques proposés pour chacun des deux problèmes. Ces modèles sont sous forme de programmes linéaires à variables mixtes caractérisés par l'indexation de la couleur au niveau de leurs variables décisionnelles.
La section 5 s'intéresse à la résolution des problèmes par une méthode exacte basée sur la séparation et évaluation, où une description générale de ce type de méthodes est rappelée tout en précisant la force et la limite de leurs principales composantes. Ensuite, nous montrons comment ces composantes sont adaptées à la résolution des problèmes de coloration de graphes mixtes étudiés.
Enfin dans la section 6, une approche métaheuristique basée sur l'algorithme tabu search est proposée avec une description de son fonctionnement et de son adaptation pour la résolution des deux problèmes de coloration de graphes mixtes.

## 3.2 Bornes inférieures

Nous proposons dans cette section une borne inférieure ($LB_\chi$) pour le nombre chromatique et une borne inférieure ($LB_\sigma$) pour la somme des couleurs des extrémités terminales des chemins de longueur maximale.

### 3.2.1 La borne inférieure $LB_\chi$ pour $\chi$

Avant d'introduire la borne inférieure $LB_\chi$ proposée pour le nombre chromatique $\chi$, nous considérons les notations suivantes :

- $H^j = (V, E^j, A)$ : le graphe partiel de G engendré par l'ensemble des arcs A et le sous-ensemble d'arêtes $E^j$, où $G^j = (V^j, E^j, \emptyset)$ est une clique maximale.
- $in(v)$ : inpath d'un sommet $v$ correspondant à la longueur du chemin de longueur maximale possédant $v$ comme extrémité terminale.
- $out(v)$ : outpath d'un sommet $v$ correspondant à la longueur du chemin de longueur maximale possédant $v$ comme extrémité initiale.

**Proposition 1.**
$\chi(G) \geq LB_\chi = \max_{j=1,\ldots,m} \chi(H^j)$.

*Démonstration.*
Il est trivial de constater que le nombre chromatique d'un graphe mixte est supérieur ou égal au nombre chromatique de chacun de ses graphes partiels. Alors, $\chi(G) \geq \max_{j=1,...,m} \chi(H^j)$. □

**Proposition 2.**
L'algorithme 3.1 colore de manière optimale chaque graphe partiel $H^j = (V, E^j, A)$, $j \in 1, ..., m$.

**Algorithme 3.1** M-Coloration

**Entrée :** Un graphe partiel $H^j = (V, E^j, A)$, $j \in 1, ..., m$
**Sortie :** Une coloration optimale des sommets
S : un ensemble de tous les chemins de longueur maximale dans $H^j$ ($S = \bigcup_{i=1}^{j=n} G_i$).
**tant que** $S \neq \emptyset$ **faire**
(1) Sélectionner dans S un chemin $G_i$ possédant le plus long sous-chemin défini à partir du sommet appartenant à la clique maximale.
(2) $S \leftarrow S \setminus \{G_i\}$.
(3) Affecter la plus petite couleur disponible à chaque sommet de $G_i$.
**fin tant que**

*Démonstration.*
Nous démontrons la proposition par l'absurde. Soient $v_i$ et $w_i$ les deux sommets du chemin de longueur maximale $G_i$, avec $v_i$ appartenant à la clique maximale considérée (i.e. $v_i \in \{V_i \cap V^j\}$) et $w_i$ l'extrémité terminale de $G_i$, où $i \in \{1, ..., n\}$.
Pour chaque chemin de longueur maximale $G_i, i \in \{1, ..., n\}$, nous avons :

$$\psi(w_i) \leq \chi(H^j) \qquad \text{et} \qquad \psi(w_i) = \text{out}(v_i) + \psi(v_i) \qquad (i)$$

Supposons qu'il existe une coloration optimale $\gamma$ dans laquelle il existe un chemin de longueur maximale $G_l$ vérifiant :

$$\text{out}(v_l) = \max_{i=1,...,n} \text{out}(v_i) \qquad (ii)$$

Nous supposons aussi, que le sommet $v_l$ de $G_l$ n'est pas coloré à la première plus petite couleur disponible, et qu'il existe un autre chemin $G_k$ possédant un sommet $v_k$ imposé d'être coloré juste avant $v_l$. Nous obtenons alors :

$$\psi(v_k) < \psi(v_l) \qquad (iii)$$

$$\text{out}(v_k) < \text{out}(v_l) \qquad (iv)$$

Il est ainsi évident que, $\psi(w_k) < \psi(w_l)$ (cf. (i), (iii) et (iv)). Par conséquent :

$$\chi(H^j) = \begin{cases} \psi(w_l) & \text{si } \chi \text{ est associé au sommet } w_l \\ \psi(w_z) & \text{si } \chi \text{ est associé au sommet } w_z, \text{ où } z \in \{1, .., n\} \setminus \{l, k\} \end{cases}$$

Maintenant, dans la coloration optimale $\gamma$, échangeons la couleur de $(v_k)$ et $(v_l)$. Nous obtenons une nouvelle coloration $\gamma'$ tel que :

1. si dans la coloration $\gamma$ on a : $\chi(H^j) = \psi(w_l)$ alors dans la coloration $\gamma'$ on obtient : $\chi(H^j) = \psi'(w_l) = \psi(w_l) - 1$.
2. si dans la coloration $\gamma$ on a : $\chi(H^j) = \psi(w_z)$, où $z \in \{1,..,n\} \setminus \{l,k\}$, alors dans la coloration $\gamma'$ on obtient : $\chi(H^j) = \psi'(w_z) = \psi(w_z)$.

On a obtenu une 'meilleure' coloration $\gamma'$, ce qui contredit l'optimalité de la coloration $\gamma$. □

**Exemple 3.1.**
Considérons le problème d'ordonnancement $\pi_{3.1}$ présenté dans le tableau 3.1. Ce problème comporte 4 jobs $\{J_1, J_2, J_3, J_4\}$ à ordonnancer sur 4 machines $\{M_1, M_2, M_3, M_4\}$. Chaque job est constitué de 4 opérations, avec un ordre de passage sur les différentes machines précis comme suit :

TABLEAU 3.1 – Les données du problème d'ordonnancement $\pi_{3.1}$

| Job | $1^{er}$ passage | | $2^{ième}$ passage | | $3^{ième}$ passage | | $4^{ième}$ passage | |
|---|---|---|---|---|---|---|---|---|
| | Machine | Opération | Machine | Opération | Machine | Opération | Machine | Opération |
| $J_1$ | $M_3$ | $O_{11}$ | $M_4$ | $O_{12}$ | $M_1$ | $O_{13}$ | $M_2$ | $O_{14}$ |
| $J_2$ | $M_3$ | $O_{21}$ | $M_4$ | $O_{22}$ | $M_1$ | $O_{23}$ | $M_2$ | $O_{24}$ |
| $J_3$ | $M_3$ | $O_{31}$ | $M_1$ | $O_{32}$ | $M_2$ | $O_{33}$ | $M_4$ | $O_{34}$ |
| $J_4$ | $M_4$ | $O_{41}$ | $M_2$ | $O_{42}$ | $M_3$ | $O_{43}$ | $M_1$ | $O_{44}$ |

Le graphe mixte G associé à ce problème est présenté dans la Figure 3.1. Ce graphe est composé de 16 sommets, chacun représentant une opération du problème, et de 4 chemins de longueur maximale représentant les opérations des jobs à exécuter dans l'ordre, et de 4 cliques maximales représentant les opérations à exécuter par la même machine.

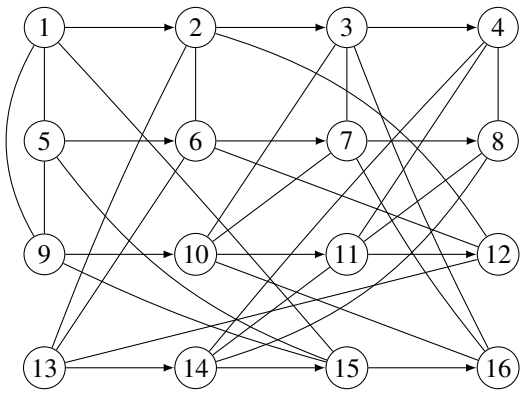


FIGURE 3.1 – Graphe mixte associé au problème $\pi_{3.1}$

Les graphes partiels $H^1 = (V, E^1, A)$, $H^2 = (V, E^2, A)$, $H^3 = (V, E^3, A)$ et $H^4 = (V, E^4, A)$ sont respectivement présentés dans les sous-figures (a), (b), (c) et (d) de la figure 3.2.

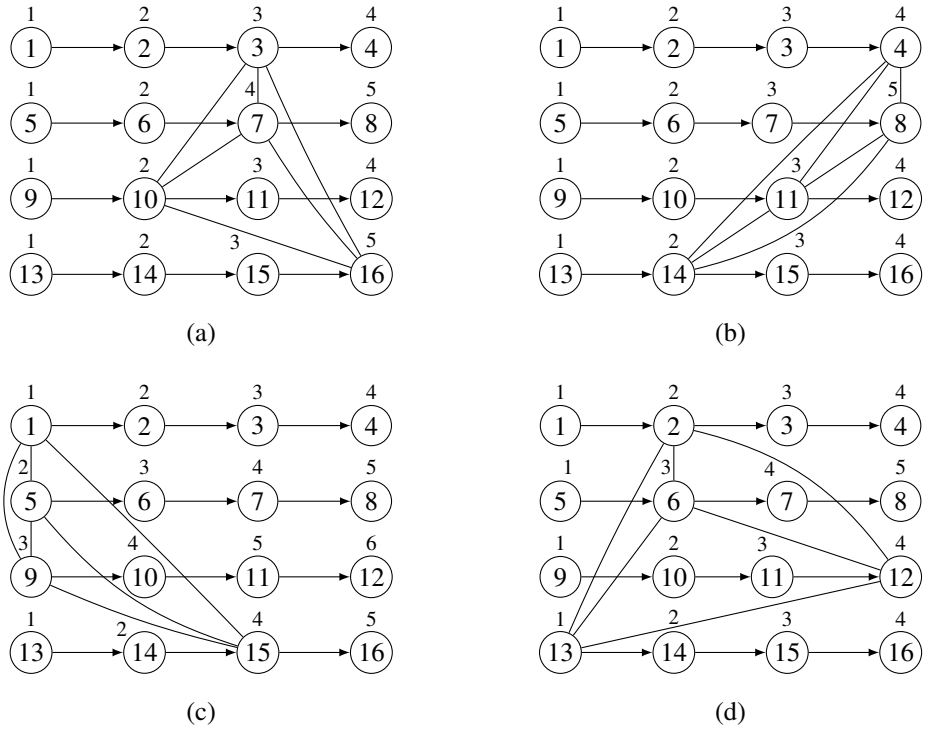


FIGURE 3.2 – Graphes partiels associés au problème $\pi_{3.1}$

Nous appliquons l'algorithme 3.1 pour la coloration de ces graphes partiels, et en se basant sur les propositions 1 et 2, la borne inférieure $LB_\chi$ pour le nombre chromatique $\chi$ du graphe mixte présenté dans la figure 3.1 est obtenue comme suit :

$$LB_\chi = \max_{i=1,\dots,4} \chi(H^i) = \max(5,5,6,5) = 6$$

Pour comparer la valeur de la borne proposée avec celle de la littérature, rappelons la borne de Sotskov et al. (2001) décrite dans le chapitre précédent (cf. chap2, §2.4.3, eq 2.2) :

$$LB_{\chi 1} = \max_{M_k \in M} \{\min_{j_d \in J}(h_k^d) + |V^k| + \min_{j_d \in J}(t_k^d)\}$$

où, $h_k^d$ (resp. $t_k^d$) est le nombre de sommets précédant (resp. succédant) un sommet appartenant à la fois à la clique maximale $k$ et au chemin de longueur maximale $d$.

Pour calculer la valeur de $LB_{\chi 1}$ associée au graphe mixte G de la figure 3.1, nous procédons pour chaque valeur de $k$ comme suit :

k=1 :$\min_{j_d \in J}(h_1^d) + |V^1| + \min_{j_d \in J}(t_1^d) = \min(2,2,1,3) + 4 + \min(1,1,2,0) = 5$
k=2 :$\min_{j_d \in J}(h_2^d) + |V^2| + \min_{j_d \in J}(t_2^d) = \min(3,3,2,1) + 4 + \min(0,0,1,2) = 5$
k=3 :$\min_{j_d \in J}(h_3^d) + |V^3| + \min_{j_d \in J}(t_3^d) = \min(0,0,0,2) + 4 + \min(3,3,3,1) = 5$
k=4 :$\min_{j_d \in J}(h_4^d) + |V^4| + \min_{j_d \in J}(t_4^d) = \min(1,1,3,0) + 4 + \min(2,2,0,3) = 4$

Ainsi, la borne inférieure $LB_{\chi 1}$ pour le nombre chromatique $\chi$ du graphe mixte présenté dans la figure 3.1 est :

$$LB_{\chi 1} = \max(5, 5, 5, 4) = 5$$

### 3.2.2 La borne inférieure $LB_\sigma$ pour $\sigma$

Dans cette sous-section, nous proposons une borne inférieure pour $\sigma$. Nous montrons que l'algorithme 3.1 qui définit une borne inférieure pour $\chi$ peut être adapté pour le calcul d'une borne inférieure pour $\sigma$ (en considérant l'objectif $\sigma$ au lieu de $\chi$ et en modifiant l'ordre de sélection des sommets pour leur assigner une couleur). Ainsi, l'algorithme 3.2 est proposé pour le calcul de cette borne.
Notons qu'afin que cette borne soit valide, nous imposons pour chaque clique maximale de ne pas avoir plus d'un sommet de chaque chemin de longueur maximale.

**Proposition 3.**
$\sigma(G) \geq LB_\sigma = \max_{j=1,...,m} \sigma(H^j)$.

*Démonstration.*
Il est trivial de constater que la somme des couleurs d'un sous-ensemble de sommets d'un graphe est supérieure ou égale à la somme des couleurs des sommets de chacun de ses graphes partiels.
Ainsi, $\sigma(G) \geq \max_{j=1,...,m} \sigma(H^j)$. □

**Proposition 4.**
En considérant le critère $\sigma$, l'algorithme 3.2 colore de manière optimale chaque graphe partiel $H^j = (V, E^j, A)$, $j \in 1, ..., m$.

**Algorithme 3.2** S-Coloration

**Entrée :** Un graphe partiel $H^j = (V, E^j, A)$, $j \in 1, ..., m$
**Sortie :** Une coloration optimale des sommets
S : un ensemble de tous les chemins de longueur maximale dans $H^j$ ($S = \bigcup_{i=1}^{j=n} G_i$).
**tant que** $S \neq \emptyset$ **faire**
(1) Sélectionner dans S un chemin $G_i$ possédant le plus court sous-chemin défini à partir de l'extrémité initiale de $G_i$ jusqu'au sommet appartenant à la clique maximale.
(2) $S \leftarrow S \setminus \{G_i\}$.
(3) Affecter la plus petite couleur disponible à chaque sommet de $G_i$.
**fin tant que**

*Démonstration.*
Notons $v_i$ et $w_i$ les deux sommets du chemin de longueur maximale $G_i$, avec $v_i$ appartenant à la clique maximale considérée (i.e. $v_i \in \{V_i \cap V^j\}$) et $w_i$ l'extrémité terminale de $G_i$, où $i \in \{1, ..., n\}$.

Pour chaque chemin de longueur maximale $G_i, i \in \{1, ..., n\}$, nous avons :

$$\psi(w_i) = \text{out}(v_i) + \psi(v_i) \qquad (i)$$

Nous considérons une coloration optimale $\gamma$ dans laquelle il existe un chemin de longueur maximale $G_l$ vérifiant :

$$\text{in}(v_l) = \min_{i=1,...,n} \text{in}(v_i) \qquad (ii)$$

Supposons que le sommet $v_l$ de $G_l$ n'est pas coloré à la première plus petite couleur disponible, et qu'il existe un autre chemin $G_k$ possédant un sommet $v_k$ imposé d'être coloré avant $v_l$.
Nous obtenons donc :

$$\psi(v_k) < \psi(v_l) \qquad (iii)$$

$$\text{in}(v_k) > \text{in}(v_l) \qquad (iv)$$

Le coût associé à la coloration $\gamma$ est :

$$\sum_{i \in \{1..,n\}} \psi(w_i) = \psi(w_k) + \psi(w_l) + \sum_{i \in \{1..,n\} \setminus \{k,l\}} \psi(w_i) \qquad (v)$$

Maintenant dans la coloration optimale $\gamma$, modifions la coloration de $(v_l)$ et $(v_k)$ en imposant à ce que $(v_l)$ soit coloré avant $(v_k)$ (i.e. sans modifier la coloration des autres sommets).

En se basant sur la relation (i), nous obtenons donc une nouvelle coloration $\gamma'$ vérifiant :

1. $\psi'(w_k) = \psi(w_k) + 1$ et $\psi'(w_l) = \psi(w_l) - 1$ (dans le cas où $\psi(v_k) = in(v_k)$ et la couleur $in(v_k) - 1$ est attribuée à un des sommets de la clique maximale).
2. $\psi'(w_k) = \psi(w_k) + 1$ et $\psi'(w_l) = \psi(w_l) - 2$ (dans le cas où $\psi(v_k) > in(v_k)$ et la couleur $in(v_k) - 1$ n'est pas attribuée à un des sommets de la clique maximale).

Etant donné que dans la nouvelle coloration $\gamma'$ tous les sommets autres que ceux appartenant aux chemins des sommets $(v_l)$ et $(v_k)$ ont gardé leur couleurs, alors le coût de la coloration de $\gamma'$ est :

$$\sum_{i \in \{1..,n\}} \psi'(w_i) = \psi'(w_k) + \psi'(w_l) + \sum_{i \in \{1..,n\} \setminus \{k,l\}} \psi(w_i) \qquad (vi)$$

Par conséquent :

$$\sum_{i \in \{1..,n\}} \psi'(w_i) = \begin{cases} \psi(w_k) + \psi(w_l) + \sum_{i \in \{1..,n\} \setminus \{k,l\}} \psi(w_i) \\ \psi(w_k) + \psi(w_l) - 1 + \sum_{i \in \{1..,n\} \setminus \{k,l\}} \psi(w_i) \end{cases}$$

On a obtenu une 'meilleure' coloration $\gamma'$, cela contredit l'optimalité de la coloration $\gamma$. □

**Exemple 3.2.**
Revenons au problème de l'exemple 3.1. Cet exemple représente un cas particulier dans lequel les chemins ont la même longueur maximale et, par conséquent, il en résulte le même ordre de sélection de chemins au niveau des deux algorithmes 3.2 et 3.1.
En appliquant l'algorithme 3.2 pour colorer les graphes partiels considérés, nous obtenons la coloration présentée dans la figure 3.2.
Selon les propositions 3 et 4, la borne inférieure $LB_\sigma$ pour σ du graphe mixte présenté dans la figure 3.1 est détaillée comme suit :

$$LB_\sigma = \max_{i=1,\dots,4} \sigma(H^i) = \max(18, 17, 20, 17) = 20$$

Nous comparons maintenant $LB_\sigma$ avec la borne inférieure $LB_{\sigma 1}$ d'Al-Anzi et al. (2006) détaillée dans le chapitre précédent (cf. chap2, §2.4.3, eq 2.6) :

$$LB_{\sigma 1} = \max_{M_k \in M} \{n \min_{j_d \in J}(h_k^d) + \frac{n(n+1)}{2} + n \min_{j_d \in J}(t_k^d)\}$$

Le calcul de la valeur de $LB_{\sigma 1}$ associé au graphe mixte G de la figure 3.1 est détaillé comme suit :

k=1 : $n \min_{j_d \in J}(h_1^d) + \frac{n(n+1)}{2} + n \min_{j_d \in J}(t_1^d) = 4\min(2,2,1,3) + 10 + 4\min(1,1,2,0) = 14$
k=2 : $n \min_{j_d \in J}(h_2^d) + \frac{n(n+1)}{2} + n \min_{j_d \in J}(t_2^d) = 4\min(3,3,2,1) + 10 + 4\min(0,0,1,2) = 14$
k=3 : $n \min_{j_d \in J}(h_3^d) + \frac{n(n+1)}{2} + n \min_{j_d \in J}(t_3^d) = 4\min(0,0,0,2) + 10 + 4\min(3,3,3,1) = 14$
k=4 : $n \min_{j_d \in J}(h_4^d) + \frac{n(n+1)}{2} + n \min_{j_d \in J}(t_4^d) = 4\min(1,1,3,0) + 10 + 4\min(2,2,0,3) = 10$

Donc, la borne inférieure $LB_{\sigma 1}$ pour le nombre chromatique σ du graphe mixte présenté dans la figure 3.1 est :

$$LB_{\sigma 1} = \max(14, 14, 14, 10) = 14$$

## 3.3 Bornes supérieures

Dans la littérature, nombreux sont les travaux qui utilisent l'algorithme DSATUR comme borne supérieur pour le nombre chromatique d'un graphe non orienté (Méndez-Díaz and Zabala, 2006), où DSATUR est un algorithme très connu proposé par Brélaz and Zabala (1979) pour la coloration de graphes non orientés.

Dans cette section nous proposons un algorithme (M-Dsat) (cf. algorithme 3.3) pour la coloration de graphes mixtes. Cet algorithme qui est une adaptation de l'algorithme Dsatur, fournit une borne supérieure ($UB_\chi$) pour le nombre chromatique (χ). Nous précisons que cet algorithme est aussi utilisé pour déterminer une borne supérieure $UB_\sigma$ pour la somme des couleurs des extrémités terminales des chemins de longueur maximale (σ).

Considérons $N(v)$ : l'ensemble des voisins du sommet $v$, $N_c(v)$ : le nombre de couleurs

différentes assignées à N($v$) et Pred($v$) : le prédécesseur du sommet $v$, (i.e. (Pred($v$), $v$) est un arc).

Pour chaque sommet $v$, son degré de saturation M-Dsat($v$) est défini comme suit :

$$\text{M-Dsat}(v) = \begin{cases} -1 & \text{si Pred}(v) \text{ est non coloré} \\ \text{N}_c(v) & \text{sinon.} \end{cases}$$

**Algorithme 3.3** M-Dsat

**Donnée :** G = (V, E, A) : un graphe mixte
**Sortie :** (UB) : une borne supérieure pour le critère de coloration du graphe mixte G
(1) Ordonner les sommets par ordre croissant de leur inpath
(2) Colorer le premier sommet par la couleur 1
(3) Choisir un sommet $v$ avec un degré maximal de saturation mixte (M-Dsat)
**si** il y a égalité **alors**
(4) Choisir un sommet possédant le plus petit inpath dans le sous-graphe non coloré
**fin si**
(5) Colorer le sommet choisi avec la plus petite couleur possible
**si** tous les sommets sont colorés **alors**
(6) Retourner la borne supérieure obtenue (UB) et fin
**sinon**
(7) Aller à 3
**fin si**

## 3.4 Modélisation mathématique

Nous allons passer à présent à la modélisation mathématique des deux problèmes de coloration traités dans le cadre de nos travaux. Rappelons tout d'abord, comme nous l'avons mentionné au chapitre 2, qu'une modélisation sous forme de programme linéaire utilisant des contraintes disjonctives engendre une explosion du nombre de cas à traiter séparément lorsque le nombre de disjonctions issues de ce type de contraintes augmente. Pour éviter l'utilisation des contraintes disjonctives, les modèles que nous proposons sont donnés sous forme de programmes linéaires à variables mixtes (Mixed Integer Linear Program MILP) caractérisés par l'indexation de la couleur au niveau des variables binaires de décision.

Le premier modèle $\text{MILP}_\chi$ modélise le problème de coloration de graphes mixtes avec comme critère le nombre chromatique $\chi$, alors que le second $\text{MILP}_\sigma$ modélise le problème de coloration de graphes mixtes minimisant la somme des couleurs des extrémités terminales des chemins de longueur maximale $\sigma$. Notons que dans ces modèles, le nombre de variables utilisées est aussi important que le résultat du produit entre le nombre de sommets et le nombre de couleurs à utiliser. Afin de mettre en oeuvre ces modèles avec un nombre fini de variables, il est nécessaire de limiter le nombre de couleurs en utilisant une borne supérieure pour le nombre chromatique.

### 3.4.1 Modélisation du problème de coloration de graphes mixtes avec le critère χ

Dans ce qui suit, nous présentons le modèle mathématique proposé pour la coloration de graphes mixtes avec comme critère le nombre chromatique χ ($MILP_\chi$), en commençant par décrire les différents paramètres, les variables de décision, les contraintes et la fonction d'objectif à optimiser.

#### 3.4.1.1 Paramètres et indices

$i, h$ : indices des chemins de longueur maximale $i$, $h = 1,..., n$

$j$ : indice relatif à l'ordre d'un sommet dans son chemin de longueur maximale $G_i = (V_i, \emptyset, A_i)$, $j = 1, ..., n_i$

$l$ : indice de la clique maximale $l = 1,..., m$

$k$ : une couleur $k$=1,...,UB

$v_{ij}$ : le $j^{ième}$ sommet du chemin de longueur maximale $G_i = (V_i, \emptyset, A_i)$

#### 3.4.1.2 Variables de décision

Y : une variable bornée dénotant le nombre de couleurs utilisées

$$X_{ijk} = \begin{cases} 1 & \text{si la couleur } k \text{ est assignée au sommet } v_{ij} \\ 0 & \text{sinon} \end{cases}$$

#### 3.4.1.3 Contraintes

Nous précisons que le nombre de couleurs d'une solution optimale ne peut dépasser la borne supérieure du nombre chromatique UB. Nous pouvons donc limiter notre attention qu'aux solutions impliquant un nombre de couleurs inférieur à UB. Ainsi, les contraintes (3.1) exigent à chaque sommet $v_{ij} \in V$ de recevoir une seule couleur $k \in \{1, ..., UB\}$.

$$\sum_{k \in \{1,...,UB\}} X_{ijk} = 1 \qquad \forall i \in \{1, ..., n\}, \forall j \in \{1, ..., n_i\} \tag{3.1}$$

Au niveau de chaque clique maximale $G^l = (V^l, E^l, \emptyset)$ ($l \in \{1, ..., m\}$), les contraintes (3.2) imposent à chaque couleur $k \in \{1, ..., UB\}$ d'être attribuée qu'à au plus un sommet $v_{ij} \in V^l$.

$$\sum_{v_{ij} \in V^l} X_{ijk} \leq 1 \qquad \forall k \in \{1, ..., UB\}, \forall l \in \{1, ..., m\} \tag{3.2}$$

Les contraintes (3.3) récupèrent la couleur de l'extrémité terminale de chaque chemin de longueur maximale, et les obligent à être inférieure à la variable de décision Y.

$$\sum_{k \in \{1,...,UB\}} k.X_{in_ik} \leq Y \qquad \forall i \in \{1, ..., n\} \tag{3.3}$$

Dans chaque chemin de longueur maximale, les couleurs des deux extrémités de chaque arc sont récupérées par les contraintes (3.4), puis la couleur de l'extrémité initiale de chaque arc est forcée d'être strictement inférieure à la couleur de son extrémité terminale.

$$\sum_{k\in\{1,...,\mathrm{UB}\}} k.\mathrm{X}_{i(j-1)k} + 1 \leq \sum_{k\in\{1,...,\mathrm{UB}\}} k.\mathrm{X}_{ijk} \qquad \forall i \in \{1,...,n\}, \forall j \in \{2,...,n_i\} \tag{3.4}$$

Les contraintes (3.5) imposent à ce que la variable Y soit bornée ente la borne inférieure (LB) et la borne supérieure (UB).

$$\mathrm{LB} \leq \mathrm{Y} \leq \mathrm{UB} \qquad \mathrm{Y} \in \mathrm{N} \tag{3.5}$$

#### 3.4.1.4 Fonction objectif

La fonction objective (3.6) minimise le nombre de couleurs utilisées.

$$\text{Minimize} \quad \mathrm{Y} \tag{3.6}$$

Ainsi le programme linéaire à variables mixtes $\mathrm{MILP}_\chi$ proposé pour la résolution du problème de coloration de graphes mixtes est comme suit :

$$\mathrm{MILP}_\chi = \begin{cases} \text{Minimize} \quad \mathrm{Y} & & & & \\ \displaystyle\sum_{k\in\{1,...,\mathrm{UB}\}} \mathrm{X}_{ijk} & = & 1 & \forall i \in \{1,...,n\}, \forall j \in \{1,...,n_i\} & (3.1) \\ \displaystyle\sum_{v_{ij}\in \mathrm{V}^l} \mathrm{X}_{ijk} & \leq & 1 & \forall k \in \{1,...,\mathrm{UB}\}, \forall l \in \{1,...,m\} & (3.2) \\ \displaystyle\sum_{k\in\{1,...,\mathrm{UB}\}} k.\mathrm{X}_{in_ik} & \leq & \mathrm{Y} & \forall i \in \{1,...,n\} & (3.3) \\ \displaystyle\sum_{k\in\{1,...,\mathrm{UB}\}} k.\mathrm{X}_{i(j-1)k} + 1 & \leq & \displaystyle\sum_{k\in\{1,...,\mathrm{UB}\}} k.\mathrm{X}_{ijk} & \forall i \in \{1,...,n\}, \forall j \in \{2,...,n_i\} & (3.4) \\ \mathrm{LB} \leq \mathrm{Y} \leq \mathrm{UB} & & & \mathrm{Y} \in \mathrm{N} & (3.5) \\ \mathrm{X}_{ijk} \in \{0,1\} & & & \forall i \in \{1,...,n\}, \forall j \in \{1,...,n_i\}, \forall k \in \{1,...,\mathrm{UB}\} & \end{cases}$$

### 3.4.2 Modélisation du problème de coloration de graphes mixtes avec le critère σ

Pour le problème de coloration de graphes mixtes optimisant la somme des couleurs des extrémités terminales des chemins de longueur maximale, le modèle mathématique ($\mathrm{MILP}_\sigma$) est caractérisé par les mêmes paramètres, variables binaires de décision, et par les contraintes (3.1), (3.2) et (3.4). Seule la fonction objectif change et est donnée comme suit :

$$\text{Minimize} \sum_{i\in\{1,...,n\}} \sum_{k\in\{1,...,\mathrm{UB}\}} k\mathrm{X}_{in_ik} \tag{3.7}$$

Ainsi le programme linéaire à variables mixtes proposé pour la résolution du problème de coloration de graphes mixtes en considérant le critère σ ($MILP_\sigma$) est défini comme suit :

$$
MILP_\sigma = \begin{cases}
\text{Minimize} \sum_{i\in\{1,...,n\}} \sum_{k\in\{1,...,UB\}} k X_{in_ik} & & & \\
\sum_{k\in\{1,...,UB\}} X_{ijk} = 1 & & \forall i \in \{1,...,n\}, \forall j \in \{1,...,n_i\} & (3.1) \\
\sum_{v_{ij}\in V^l} X_{ijk} \leq 1 & & \forall k \in \{1,...,UB\}, \forall l \in \{1,...,m\} & (3.2) \\
\sum_{k\in\{1,...,UB\}} kX_{i(j-1)k}+1 \leq \sum_{k\in\{1,...,UB\}} kX_{ijk} & & \forall i \in \{1,...,n\}, \forall j \in \{2,...,n_i\} & (3.4) \\
X_{ijk} \in \{0,1\} & & \forall i \in \{1,...,n\}, \forall j \in \{1,...,n_i\}, \forall k \in \{1,...,UB\} &
\end{cases}
$$

### 3.4.3 Taille des deux modèles

Nous donnons ici une idée du nombre de variables et de contraintes requis pour chaque modèle, le modèle $MILP_\chi$ est composé de $UB\sum_{i=1}^{i=n}(n_i)$ variables binaires, d'une seule variable entière, et de $2\sum_{i=1}^{i=n}(n_i) + mUB + 1$ contraintes.
Cependant, le modèle $MILP_\sigma$ est composé de $UB\sum_{i=1}^{i=n}(n_i)$ variables binaires et de $2\sum_{i=1}^{i=n}(n_i) + mUB - n$ contraintes.
Ces contraintes sont détaillées comme suit :

- $\sum_{i=1}^{i=n}(n_i)$ contraintes de type (3.1)
- $mUB$ contraintes de type (3.2)
- $n$ contraintes de type (3.3)
- $\sum_{i=1}^{i=n}(n_i - 1)$ contraintes de type (3.4)
- 1 contrainte de type (3.5)

## 3.5 Méthodes par séparation et évaluation

Dans cette section, nous nous intéressons à l'utilisation d'une autre méthode exacte basée sur la séparation et l'évaluation (branch and bound) pour la résolution de nos problèmes. Rappelons que ce type de méthodes, proposée initialement par Land and Doig (1960), utilise une stratégie de recherche arborescente pour énumérer implicitement des solutions possibles au problème d'optimisation combinatoire considéré, et applique ensuite des règles d'élagage pour éliminer les régions de l'espace de recherche qui ne peuvent pas conduire à des meilleures solutions.

Le principe de ces méthodes consiste à alterner une étape dite de séparation et une autre dite d'évaluation jusqu'à l'obtention d'une solution optimale.

De manière générale, ces méthodes sont mises en oeuvre en stockant des solutions partielles (associées aux sous-problèmes des noeuds) dans une structure arborescente. Les noeuds inexplorés de l'arbre de solutions génèrent des fils en divisant l'espace de solutions en petites régions pouvant être résolues de façon récursive (séparation). Des règles sont ensuite utilisées pour éliminer les régions de l'espace de recherche qui ne contiennent pas de solutions optimales (évaluation). Une fois l'arbre est entièrement exploré, la meilleure solution trouvée dans la recherche est retournée.

Signalons que trois composantes importantes de ce type de méthodes peuvent avoir des impacts significatifs sur leur performance. Ces composantes sont : la stratégie de recherche (i.e. l'ordre dans lequel les noeuds de l'arbre sont explorés), la stratégie de séparation (i.e. comment l'espace de solution est partitionné pour produire de nouveaux sous-problèmes dans l'arbre ?) et les règles d'élagage (i.e. quelles sont les règles à utiliser pour empêcher l'exploration des régions contenant des solutions non optimales ?).

### 3.5.1 Stratégies de recherche

La stratégie de recherche dans une méthode branch and bound consiste à déterminer l'ordre d'exploration des noeuds de l'arbre. Cet ordre de sélection des noeuds influe de manière significative sur l'efficacité de la méthode en terme de temps de résolution et de taille de mémoire utilisée. Plus de détails concernant cette idée peuvent être retrouvés en consultant les travaux d'Ibaraki (1976) qui présentent une étude comparative entre un certain nombre de stratégies de recherche.

Dans la suite, nous présentons une description des stratégies de recherche les plus utilisées dans la littérature.

#### 3.5.1.1 Recherche en profondeur (DFS)

Le parcours en profondeur DFS (Depth First Search) visite un noeud $v$ de l'arbre de solutions, puis visite un des noeuds voisins non visités de $v$. Si tous les noeuds-fils de $v$ sont visités, DFS choisit l'un des noeuds-fils du noeud précédent, et ainsi de suite jusqu'a ce que tous les noeuds de l'arbre soient visités (cf. algorithme 3.4).

Dans la mise en oeuvre de l'algorithme branch and bound, la stratégie de recherche basée sur DFS ne stocke que le chemin allant de la racine de l'arbre jusqu'au noeud courant.
Toutefois, les implémentations naïves de DFS n'utilisent aucune information concernant les caractéristiques du problème, ce qui signifie que le processus de recherche peut consacrer beaucoup de temps d'exploration dans des régions pauvres de l'espace de recherche. Pour faire face à ce problème, il est préférable de considérer tous les noeuds-fils du noeud courant, et sélectionner celui possédant la meilleure borne inférieure associée au sous-problème qui contient. Selon Scholl and Klein (1999), l'utilisation de la borne inférieure dans le choix

du prochain noeud permet d'explorer plus intelligemment l'arbre de solutions et d'éviter l'utilisation accrue de la mémoire.

Ce parcours peut être efficacement implémenté en maintenant la liste des noeuds inexplorés dans une pile. L'algorithme supprime le premier élément de la pile pour choisir le noeud suivant à explorer, et lorsque les noeuds-fils sont générés à la suite d'une séparation, ils seront forcément insérés au début de la pile. Ainsi, le prochain noeud qui sera exploré est le noeud généré le plus récemment.

**Algorithme 3.4** DFS(T,$v$)

**Donnée :** Un arbre T et un noeud de départ $v$
**début**
(1) Empiler($v$)
**tant que** PILE est non vide **faire**
(2) $u \longleftarrow$ Dépiler(PILE)
(3) Visiter $u$
**pour tout** $s \in$ Noeuds_fils($u$) **faire**
(4) Empiler ($s$)
**fin pour**
**fin tant que**
**fin**

#### 3.5.1.2 Recherche en largeur (BFS)

Le parcours en largeur BFS (Breadth-first search) visite les noeuds de l'arbre par ordre de largeur en commençant par explorer le noeud racine, puis tous ses noeuds-fils. Ensuite pour chaque noeuds-fils, il explore ses noeuds-fils, et ainsi de suite. Son implémentation utilise une structure de file d'attente, premier arrivé premier servi, où le noeud d'origine est placé en premier dans une file d'attente. Ensuite à chaque itération, BFS visite le premier élément de la file et place tous ses voisins à la fin de cette dernière, et ainsi de suite jusqu'à ce que tous les éléments de la file soient visités (cf. algorithme 3.5).

**Algorithme 3.5** BFS(T,$v$)

**Donnée :** Un arbre T et un noeud de départ $v$
**début**
(1) Enfiler($v$)
**tant que** FILE est non vide **faire**
(2) $u \longleftarrow$ Défiler(File)
(3) Visiter $u$
**pour tout** $s \in$ Noeuds_fils($u$) **faire**
(4) Enfiler ($s$)
**fin pour**
**fin tant que**
**fin**

Cependant, il est clair que les solutions complètes se trouvent habituellement dans les plus longs profondeurs de l'arbre, et de ce fait BFS est généralement incapable d'exploiter les règles d'élagage qui se comparent à la meilleure solution trouvée. Par consequent, les exigences de mémoire pour BFS sont assez élevées, ce qui empêche ce parcours d'être utilisé comme stratégie de recherche dans le branch and bound. Par contre, selon Morrison et al. (2016), des exceptions pour l'utilisation efficace de BFS peuvent exister, comme le cas de présence de règles de dominances qui permettent d'exploiter efficacement la fonction d'élagage même en absence d'une bonne solution courante.

#### 3.5.1.3 Recherche par meilleure valeur (BeFS)

Le parcours par meilleure valeur BeFS (Best-first search) visite les noeuds de l'arbre en passant en premier par le noeud le plus prometteur selon une fonction heuristique d'évaluation H qui assigne une valeur pour chaque noeud de l'arbre. Puis BeFS sélectionne comme noeud suivant celui qui correspond à la meilleure valeur de H. Pour cela, BeFS stocke la liste des noeuds inexplorés dans une file de priorités qui peut être efficacement implémentée en utilisant une structure de données de type TAS [1] en considérant la valeur de H comme une clé (Cormen et al., 2009) (cf. algorithme 3.6).

L'efficacité de cette structure réside dans sa faible complexité pour la recherche du minimum (resp. maximum) et dans l'insertion ou l'ajout d'un nouvel élément (i.e. TAS permet de trouver le minimum (resp. maximum) en O(1) et insérer ou supprimer un élément en O(log n)).

**Algorithme 3.6** BeFS(T,v)

**Donnée :** Un arbre T et un noeud de départ $v$
**début**
(1) Insérer $v$ dans TAS
**tant que** TAS est non vide **faire**
(2) $u \longleftarrow$ Racine de TAS
(3) Supprimer la Racine de TAS
(4) Visiter $u$
**pour tout** $s \in$ Noeuds_fils($u$) **faire**
(5) Insérer $s$ dans TAS
**fin pour**
**fin tant que**
**fin**

L'avantage de BeFS réside dans le fait qu'il n'impose pas une exploration d'une branche avant une autre (comme le cas de DFS), et ne visite pas forcément les noeuds par ordre de largeur (comme le cas de BFS). Par conséquent, il peut souvent trouver de bonnes solutions au plus tôt comparativement aux autres stratégies de recherche (DFS et BFS). Mais dans

1. TAS : arbre binaire dont chaque noeud a une valeur supérieure (resp. inférieure) aux valeurs figurant dans ses sous-arbres. Ce type de structure, permet de retrouver directement l'élément à traiter en priorité.

certains cas, il peut devenir très long en perdant beaucoup de temps dans des régions intermédiaires de l'arbre et risque aussi de tarder pour trouver une solution optimale. Ceci peut arriver, par exemple, quand la fonction d'évaluation H assigne une même valeur à plusieurs noeuds appartenant à des branches différentes.

#### 3.5.1.4 Recherche distribuée par meilleure valeur (DBeFS)

Ce parcours, initialement proposé par Kao et al. (2009), est une hybridation entre DFS et BeFS. La combinaison de ces deux parcours permet de surmonter leurs inconvénients et d'exploiter leurs avantages. Partant du fait que BeFS est implémenté en utilisant une structure de données de type TAS pour stocker tous les noeuds inexplorés, DBeFS divise les sous-problèmes inexplorés sur une collection de TAS, appelés contours. Lorsqu'un nouveau noeud est identifié, il est inséré dans l'un des contours selon le principe que tous les noeuds de la même profondeur sont stockés dans un contour associé à cette profondeur. Pour explorer l'espace de recherche, la stratégie DBeFS visite de manière répétitive les contours non vides, en sélectionnant le meilleur noeud (selon la meilleure valeur de H) de chaque contour avant de passer au contour suivant, i.e. DBeFS explore le meilleur noeud de la profondeur 0, puis celui de la profondeur 1 et ainsi de suite, jusqu'à ce qu'il arrive au bas de l'arbre pour répéter de nouveau le même processus à partir de la profondeur 0 (cf. algorithme 3.7).

En séparant les sous-problèmes en contours, la fonction heuristique d'évaluation H assigne une valeur à chaque noeud et le noeud sera classé localement au lieu d'un classement global. Ainsi, les contours peuvent être utilisés pour regrouper l'ensemble des noeuds qui sont directement comparables. La visite répétitive des contours assure à chaque contour qu'il soit visité fréquemment, ce qui permet à diversifier la recherche et peut aider à générer des meilleures solutions.

La stratégie DBeFS peut être implémentée à l'aide d'un ensemble de TAS, où chaque TAS mémorise l'adresse des noeuds appartenant à la même profondeur de l'arbre de solutions.

**Algorithme 3.7** DBeFS(T,v)

**Donnée :** Un arbre T et un noeud de départ $v$
**début**
(1) Insérer $v$ dans le TAS correspondant
**tant que** tous les TAS sont non vide **faire**
**pour tout** $\text{TAS}_i \in$ Ensemble_TAS **faire**
(2) $u \longleftarrow$ Racine de $\text{TAS}_i$
(3) Supprimer la Racine de $\text{TAS}_i$
(4) Visiter $u$
**pour tout** $s \in$ Noeuds_fils($u$) **faire**
(5) Insérer $s$ dans $\text{TAS}_{i+1}$
**fin pour**
**fin pour**
**fin tant que**
**fin**

### 3.5.2 Stratégies de séparation

La séparation (branchement) est une procédure itérative utilisée pour le développement de l'arbre de solutions. Le choix d'une stratégie de séparation a pour objectif de déterminer comment les fils d'un noeud sont générés. Selon Morrison et al. (2016), les stratégies de branchement peuvent être classées en deux groupes principaux : stratégies de séparation binaire et stratégies de séparation non-binaire.

#### 3.5.2.1 Séparation binaire

La stratégie de séparation binaire consiste à subdiviser le sous-problème associé à un noeud en deux sous-problèmes mutuellement exclusifs. A titre d'exemple dans le problème de l'ordonnancement à une machine ($1|r_i$, $d_i$ $|\text{N}_t$), avec comme objectif la minimisation du nombre de tâches à ordonnancer en retard. La stratégie de séparation sélectionne une tâche non attribuée et crée ensuite deux branches. Dans le premier branchement, la tâche est incluse dans l'ordonnancement, alors que dans le second branchement la tâche est exclue de l'ordonnancement (Carlier, 1982).

Nous pouvons particulièrement trouver ce type de stratégies dans la résolution des programmes linéaires en nombres entiers. La stratégie de séparation binaire dans ce cas consiste à utiliser deux phases : la sélection d'une variable ou d'un ensemble de variables à séparer, puis la création de sous-problèmes fils. Le choix de la variable de séparation peut avoir un impact significatif sur la performance de l'algorithme, et de nombreuses techniques existent pour choisir les meilleures variables de branchement. Un aperçu sur les différentes stratégies de sélection des variables est largement détaillé dans (Morrison et al., 2016; Achterberg et al., 2005).

#### 3.5.2.2 Séparation non-binaire

Dans la stratégie de séparation non-binaire, la subdivision d'un sous problème associé à un noeud consiste à sélectionner un élément d'un ensemble qui énumère tous les différents cas possibles de solutions. Par exemple dans les algorithmes Branch and Bound traitant la coloration de graphes, un ensemble de sommets non colorés est souvent maintenu au niveau de chaque noeud de l'arbre. Ensuite, ces sommets non colorés sont utilisés pour définir la coloration suivante et générer plusieurs noeuds-fils.
Morrison et al. (2016) estiment qu'une stratégie de branchement binaire considère individuellement chaque noeud non exploré, ce qui crée une longue séquence de sous-problèmes. Cette longue séquence peut être contournée par la stratégie de branchement non-binaire. Néanmoins, cette dernière pose aussi un problème lorsque le nombre de branches à créer pour un sous-problème est trop élevé. Dans ce cas, la méthode pourrait se bloquer en générant des noeuds-fils à un sous-problème et risque de tarder pour passer à l'exploration de nouvelles régions de l'espace de recherche. En outre, si le nombre de fils générés est très important pour chaque sous-problème, la taille de l'arbre de solutions augmentera alors beaucoup plus rapidement.

### 3.5.3 Règles d'élagage et de domination

On dit qu'un noeud de l'arbre est élagué (stérilisé) s'il est inutile qu'il soit séparé. Tout noeud qui ne peut pas être stérilisé par une des règles d'élagage doit impérativement être exploré par une des stratégies de recherche, même dans le cas où une solution optimale est déjà rencontrée.
Dans la sous-section précédente, nous avons évoqué le problème de la dimension très élevée de l'arbre de solutions. Une façon de réduire cette taille est d'utiliser des règles d'élagage. Parmi les meilleures règles d'élagage, nous citons les bornes inférieures et les règles de dominance décrites dans la suite. Toutefois, d'autres règles et techniques permettant de réduire l'espace de recherche existent et sont généralement dédiées à la résolution des programmes linéaires en nombres entiers, à savoir les coupes et la génération de colonnes qui sont largement détaillées dans (Morrison et al., 2016).

#### 3.5.3.1 Bornes inférieures

L'utilisation de la borne inférieure constitue la règle la plus utilisée pour élaguer les noeuds contenant des sous-problèmes avec une solution estimée en utilisant une borne inférieure et dont le coût qui n'est pas meilleur que celui de la meilleure solution rencontrée. Les bornes inférieures sont calculées en relaxant quelques aspects du problème étudié. Par exemple, calculer le nombre chromatique d'un graphe G en utilisant ses sous-graphes (cf. chap3, § 3.2).

#### 3.5.3.2 Règles de dominance

Les règles de dominance sont des stratégies d'élagage spécifiques aux problèmes qui exploitent les propriétés particulières des solutions optimales. Certaines règles de dominance peuvent même éliminer certaines solutions optimales en garantissant de conserver au moins une solution optimale sans l'éliminer. Le principe de ces règles de dominance peuvent réduire considérablement l'espace de recherche.

Dans une méthode branch and bound, ces règles permettent d'élaguer les noeuds contenant des sous-problèmes dominés par un autre sous-problème. Autrement dit, si un sous-problème P1 domine un sous-problème P2, cela signifie que, pour chaque sous-solution descendante de P2, il existe une sous solution descendante de P1 qui possède au moins la même qualité. Donc, il suffit seulement d'explorer le noeud contenant P1.

### 3.5.4 Adaptation de branch and bound pour la coloration de graphes mixtes

Dans les sous section précédente nous avons décrit le fonctionnement général de la méthode branch and bound et précisé qu'elle est composée de trois composantes importantes pouvant avoir des impacts significatifs sur sa performance. La première (stratégie de recherche) a un objectif de guider rapidement la recherche vers une solution optimale, la deuxième (stratégies de séparation) vise à limiter la génération de sous-problèmes non importants et la troisième (règles d'élagage et de domination) vise à élaguer un maximum de sous-problèmes pour réduire l'espace de recherche. Nous retenons que ces trois composantes visent à alléger le problème de surdimensionnement de l'arbre de solutions tout en guidant rapidement la recherche vers une solution optimale.

Partons des objectifs de ces composantes et de l'idée que les solutions importantes sont celles contenues dans les noeuds pendants de l'arbre de solutions, alors seuls ces noeuds présentent un intérêt majeur dans les méthodes branch and bound. Il sera donc très intéressant d'avoir un mécanisme permettant de manipuler uniquement ces noeuds pendants, et d'éliminer les autres noeuds afin de libérer l'espace de stockage et de surmonter toutes les difficultés liées à l'insuffisance de mémoire.

Bien évidemment pour mettre en oeuvre ce mécanisme, il est nécessaire d'encapsuler au niveau de chaque noeud les informations nécessaires pour assurer le bon fonctionnement de la méthode sans avoir recours aux ascendants des noeuds pendants. Ensuite sur la base de cette structure, il faut que le mécanisme assure l'utilisation des algorithmes liés à la stratégie de recherche adoptée.

Considérons maintenant le problème de coloration d'un graphe mixte G, caractérisé par un ensemble de solutions réalisables S et une fonction objectif $f : \mathrm{S} \rightarrow \mathrm{N}$. Le but est de trouver une solution optimale $s^* \in \mathrm{S}$ minimisant $f$.

Pour résoudre ce problème, l'algorithme branch and bound, construit itérativement un arbre de solutions T. A chaque itération l'algorithme sélectionne un noeud $x$ de T selon la stratégie de recherche décrite dans la section 3.5.1.1 qui utilise une structure de données de type pile (PILE). L'algorithme procède comme suit :

- si le noeud $x$ comporte une solution complète et ne peut être élagué, alors l'algorithme appelle la procédure de séparation du sous-problème P associé au noeud $x$. Puis, il crée des noeuds-fils pour $x$, et stocke leur adresse dans PILE.
- si le noeud $x$ comporte une solution complète, alors l'algorithme met à jour la meilleure solution rencontrée $s_{best}$ après l'avoir comparée avec la solution courante du problème $s_p$.

Notons que le noeud $x$ est supprimé à la fin de chaque itération de l'algorithme, et que tout le processus est répété jusqu'à ce que PILE soit vide.

Le pseudo-code de la méthode est présenté dans l'algorithme 3.8, et le détail de ses principales composantes est décrit dans les sous-sections suivantes.

**Algorithme 3.8** Branch and Bound pour la coloration de graphes mixtes

**Donnée :** Un graphe G et une borne supérieure $s_{ub}$
**Sortie :** $s_{best}$
**début**
$s_{best} \leftarrow s_{ub}$
(1) Générer la racine $x_0$ de l'arbre T
(2) Insérer l'adresse de la racine dans PILE
**tant que** PILE est non vide **faire**
(3) $u \longleftarrow$ Dépiler(PILE)
**si** le noeud $x$ de l'adresse $u$ ne comporte pas une solution complète et son sous problème P ne peut être élagué **alors**
(4) Partitionner P en sous problèmes $P_1, P_2, ..., P_r$
(5) Générer $r$ noeuds-fils pour $x$, et affecter à chaque noeud une solution des sous problèmes $\{P_1, ..., P_r\}$
(6) Insérer les adresses des noeuds générés, par ordre croissant de leur borne inférieure, dans PILE
**fin si**
**si** le noeud $x$ de l'adresse $u$ comporte une solution complète **alors**
**si** $f(s_p) < f(s_{best})$ **alors**
(7) $s_{best} \leftarrow s_p$
**fin si**
**fin si**
(8) Supprimer le noeud $x$
**fin tant que**
**fin**

#### 3.5.4.1 Schéma de séparation adopté

Nous commençons par décrire le codage d'une solution d'un sous-problème associé à un noeud de l'arbre. Pour ce faire, nous nous inspirons du codage proposé par Al-Anzi et al. (2006) et de Sotskov et al. (2001) qui utilise uniquement un vecteur V d'entiers. Dans cette étude, pour le but de limiter le nombre d'accès à l'arbre pour récupérer les informations relatives à l'ordre des sommets colorés et de mémoriser les solutions partielles appartenant au noeuds ascendants, nous ajoutons à ce codage deux autres vecteurs O et S.

$$\mathrm{V} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} \qquad \mathrm{O} = \begin{pmatrix} o_1 \\ o_2 \\ \vdots \\ o_n \end{pmatrix} \qquad \mathrm{S} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_{|v|} \end{pmatrix}$$

La composante $v_i$ du vecteur V correspond à la couleur associée à un sommet du chemin $i$, et la composante $o_i$ du vecteur O correspond à l'ordre de ce sommet dans le chemin $i$. Quant au vecteur S, il est scindé en $n$ parties (où $n$ étant le nombre de chemins), et chacune des parties est composée de $n_i$ composantes (où $n_i$ étant le nombre de sommets du chemin $i$).

L'iteration initiale de la méthode branch and bound définit la racine de l'arbre, dans laquelle les trois vecteurs sont nuls. Ceci signifie qu'aucun sommet du graphe mixte n'est coloré à cette étape. Dans la première itération, l'algorithme énumère tous les cas possibles pour colorer les sommets du graphe mixte par la couleur 1 (sous réserve, évidemment, qu'une coloration ne cause aucun conflit entre les sommets d'une même arête).
Après cette énumération, l'algorithme utilise la stratégie de séparation non-binaire pour générer tous les sous solutions à stocker dans les noeuds-fils de la racine de l'arbre. Pour cela, au niveau de chaque noeud généré, toutes les composantes du vecteur O seront assignées par la valeur 1. Ensuite, chaque composante $v_i$ du vecteur V lui sera assignée la valeur 1 (si le sommet correspondant est coloré par la couleur 1), ou la valeur -1 (présence de conflits).

De manière générale, si la valeur d'une composante $o_i$ est égale au nombre des sommets du chemin $i$ et que la valeur de $v_i$ est positive, alors cela signifie que tous les sommets du chemin $i$ sont colorés, et que ce chemin ne sera donc plus considéré dans les prochaines séparations. Dans le cas inverse, les composantes des vecteurs associés aux noeuds seront mises à jour comme suit :

- si la valeur d'une composante $o_i$ est inférieure ou égale au nombre de sommets du chemin $i$, et si la valeur de $v_i$ est négative (cela signifie que le sommet correspondant a été déjà soumis à un conflit de coloration et il n'est toujours pas coloré), alors le sommet correspondant sera pris en charge dans les prochaines séparations.
- si la valeur d'une composante $o_i$ est inférieure au nombre de sommets et si la valeur de $v_i$ est positive, cela signifie que le sommet correspondant est coloré et que son

successeur sera considéré dans les prochaines séparations.

Notons qu'à chaque itération :

- Dans chaque noeud généré, les composantes du vecteur V prennent soit une valeur $h$ ou $-h$, où $h$ désigne la profondeur de l'arbre, excepté les composantes relatives à l'extrémité terminale d'un chemin, qui ne changent pas une fois assignées par une valeur positive.
- Les sommets colorés sont mémorisés au niveau de leur composante correspondante du vecteur S.
- Autre que les conditions d'élagage, les noeuds de l'arbre à ne pas séparer sont ceux qui possèdent un vecteur V positif et chaque composante du vecteur O égale au nombre des sommets du chemin correspondant.

**Exemple 3.3.**
Considérons le problème d'ordonnancement décrit dans l'exemple 3.1, où le graphe mixte associé à ce problème est présenté dans la figure 3.1. Une partie de l'arbre de solutions est décrite dans la figure 3.3.

Pour illustrer le fonctionnement de la stratégie de séparation, nous déroulons l'algorithme branch and bound, en adoptant la stratégie de recherche en profondeur (DFS), jusqu'à ce que la première solution réalisable soit trouvée. Comme précisé dans la sous-section précédente, la racine de l'arbre contient trois vecteurs nuls, signifiant qu'aucun noeud à cette étape n'est coloré. A la première itération, l'algorithme énumère tous les cas possibles pour colorer un sous ensemble des sommets par la couleur 1. D'après le graphe mixte associé au problème (cf. figure 3.1), les extrémités initiales du premier chemin (sommet 1), du deuxième chemin (sommet 5) et du troisième chemin (sommet 9) ne peuvent être à la fois colorés par la même couleur 1.
Par conséquent, pour la racine de l'arbre, l'algorithme génère trois noeuds-fils en attribuant la valeur 1 à toutes les composantes des vecteurs $O^1$, $O^2$ et $O^3$ associés à ces trois noeuds générés. Cela signifie que l'extrémité initiale de chacun des trois chemin peut être colorée à cette étape. Mais seulement une des extrémités initiales des chemins 1, 2 et 3 peut prendre une valeur positive, ce qui donne ainsi 3 cas possibles qui sont énumérés dans les 3 vecteurs $V^1$, $V^2$ et $V^3$.

Dans la seconde itération, l'algorithme passe à l'attribution de la couleur 2 aux sommets du graphe, en fonction de la coloration retenue à l'iteration précédente. Pour cela l'algorithme génère des noeuds-fils pour le cinquième noeud de l'arbre (selon la stratégie DFS). Il s'agit plus précisément de colorer le deuxième sommet du premier chemin, l'extrémité initiale du deuxième chemin, l'extrémité initiale du troisième chemin, et le deuxième sommet du quatrième chemin.

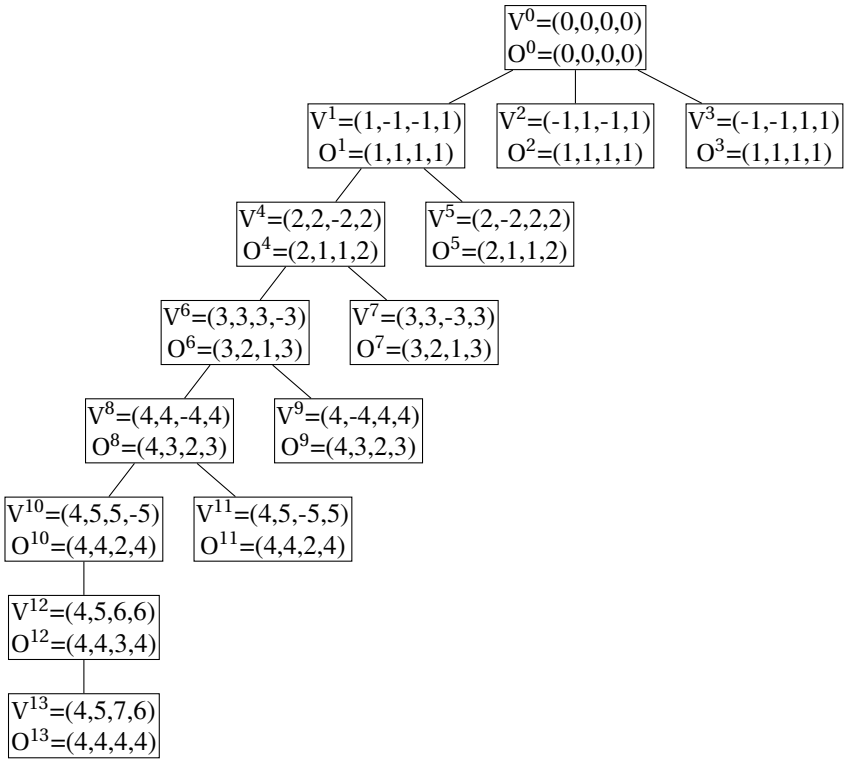


FIGURE 3.3 – Stratégie de séparation appliquée à l'arbre de solutions utilisant le parcours DFS

Mais comme les extrémités initiales du deuxième et troisième chemin sont reliées par une arête, alors la couleur 2 ne peut être à la fois assignée à ces deux sommets. Par conséquent, seuls deux noeuds-fils seront générés dans cette étape. Ce même raisonnement est répété, jusqu'à ce que la condition d'arrêt soit satisfaite (cf. figure 3.3).

#### 3.5.4.2 Stratégie de recherche adoptée

Pour la stratégie de parcours retenue dans le cadre de cette étude, nous n'avons pas exactement adopté le parcours DFS, étant donné que le nouveau noeud à traiter (dans l'algorithme développé) est le fils possédant la borne inférieure de valeur minimale (sous reserve que cette valeur soit inférieure à la borne supérieure courante). Dans le cas contraire (i.e. le noeud courant possède une solution complète), alors l'algorithme effectue un retour en arrière pour parvenir à trouver un noeud possédant au moins un frère pouvant être considéré. Mais s'il y a plusieurs frères, alors l'algorithme sélectionne celui possédant la plus petite borne inférieure.

#### 3.5.4.3 Règle d'élagage adoptée

Dans la mise en oeuvre de l'algorithme branch and bound, nous avons utilisé les bornes inférieures comme règle d'élagage. A partir de ce type de règles d'élagage, il est important de tenir compte qu'une borne inférieure rapide permet de générer un grand nombre de noeuds,

ce qui contribue à augmenter la chance d'obtenir une solution de bonne qualité mais tout en augmentant le nombre de noeuds traités par unité de temps. Alors qu'une borne inférieure plus précise possède un impact sur le nombre de noeuds élagués en consommant beaucoup de temps . Donc, il est clair que l'idéal serait d'avoir une borne inférieure rapide et plus précise. Comme ces deux critères sont dans la plupart des cas contradictoires, il suffit donc d'avoir une borne inférieure satisfaisant un bon compromis.

A chaque itération, l'algorithme dispose au niveau de chaque noeud une liste de sommets colorés et une autre liste des sommets non colorés. Pour associer à chaque sous problème du noeud une borne inférieure, l'algorithme procède à l'évaluation en considérant le critère du problème de coloration adopté.

Ainsi dans le cas du nombre chromatique ($\chi$), la borne inférieure est calculée pour le sous graphe engendré par l'ensemble des sommets non colorés et la valeur obtenue est ajoutée au nombre chromatique associé au sous graphe engendré par les sommets colorés.

Cependant dans le cas de la minimisation de la somme des couleurs des extrémités terminales des chemins de longueur maximale $\sigma$, la borne inférieure est calculée pour le sous graphe engendré par l'ensemble des sommets non colorés, sous la condition que la couleur de chaque extrémité initiale d'un chemin doit être obligatoirement supérieure à une valeur déduite de la coloration du sous graphe engendré par les sommets qui sont déjà colorés.

#### 3.5.4.4 Variantes de la méthode branch and bound développées

Dans la mise en oeuvre de la méthode branch and bound développée pour la coloration de graphes mixtes, nous avons utilisé deux bornes inférieures pour $\chi$ et deux bornes inférieures pour $\sigma$. Il en découle deux variantes de la méthode branch and bound pour chacun des deux problèmes de coloration :

- $BB1_{\chi}$ : méthode branch and bound pour la coloration de graphes mixtes avec le critère du nombre chromatique $\chi$ et utilisant la borne inférieure $LB_{\chi 1}$ de Sotskov et al. (2001).
- $BB2_{\chi}$ : méthode branch and bound pour la coloration de graphes mixtes avec le critère du nombre de chromatique $\chi$ et utilisant la borne inférieure $LB_{\chi}$ proposée.
- $BB1_{\sigma}$ : méthode branch and bound pour la coloration de graphes mixtes avec le critère $\sigma$ et utilisant la borne inférieure $LB_{\sigma 1}$ d'Al-Anzi et al. (2006).
- $BB2_{\sigma}$ : méthode branch and bound pour la coloration de graphes mixtes avec le critère $\sigma$ et utilisant la borne inférieure $LB_{\sigma}$ proposée.

## 3.6 Algorithme tabu search pour la coloration de graphes mixtes

La méthode recherche taboue (Tabu Search TS), initialement proposée par Glover (1989) et Glover (1990) est une métaheuristique largement utilisée pour la résolution des problèmes d'optimisation. Son principe consiste à guider une procédure de recherche locale à explorer l'espace de solutions au-delà de l'optimalité locale. L'algorithme TS utilise une procédure de recherche locale, où des solutions voisines sont visitées en effectuant des mouvements simples au niveau de la solution courante. Lorsqu'une meilleure solution est trouvée, elle est stockée. A chaque itération, et tant que le nombre d'itérations n'est pas atteint, ce processus est répété à partir de la meilleure solution de l'ensemble de voisinages courants.

Pour éviter l'apparition d'un cycle et de retourner à une solution déjà visitée, une liste de mouvements interdits (liste tabu) est fréquemment mise à jour. Cette liste est de taille fixe et contient les derniers mouvements effectués.

C'est dans cet ordre d'idées que nous avons adapté l'algorithme TS pour le problème de coloration de graphes mixtes dont les détails sont donnés dans la section qui suit et dans le pseudo-algorithme 3.10. Nous commençons par décrire ses principales composantes, à savoir le codage, l'évaluation d'une solution, et la structure de voisinage.

### 3.6.1 Codage d'une solution

Le choix du codage d'une solution qui permet de faciliter la mise en oeuvre des mouvements et l'évaluation des solutions est essentiel pour le succès de toute méthode de recherche locale. Dans notre contexte et afin de déterminer l'ordre d'attribution des couleurs disponibles aux différents sommets du graphe mixte, en respectant à la fois les contraintes de conflit et de precedence, nous avons adopté le codage suivant :

Considérons un vecteur d'entiers strictement positifs de $\sum_{i=1}^{i=n} n_i$ composantes qu'on note W = $[w_j, j = 1,...,|V|]$. Ce vecteur ne constitue pas une solution realisable puisque ses éléments doivent satisfaire les deux types de contraintes du problème, à savoir les contraintes de precedence imposées par les extrémités des arcs et les contraintes de conflit imposées par les extrémités des arêtes. Cependant, il peut servir de base pour déterminer une solution realisable, et ses composantes prennent leur valeur à partir de l'ensemble d'indices des chemins de longueur maximale $i = \{1,..., n\}$. Chaque indice choisi apparaît exactement $n_i$ fois dans W, où $n$ est le nombre de chemins de longueur maximale et $n_i$ est le nombre de sommets du chemin $i$.
Ainsi, la $k^{ième}$ apparence de l'indice d'un chemin dans W désigne son $k^{ième}$ sommet. Notons qu'initialement, le vecteur W est généré de manière aléatoire.

Pour l'évaluation d'une solution, nous traitons les sommets du graphe mixte dans l'ordre défini dans le vecteur W, et nous attribuons à chaque sommet du graphe la première couleur disponible tout en respectant les contraintes du problème.

**Exemple 3.4.**
Considérons de nouveau le problème d'ordonnancement décrit dans l'exemple 3.1 et représenté par le graphe mixte G de la figure 3.1. Ce graphe est composé de 4 chemins de longueur maximale, de 4 cliques de cardinalité maximale, et de 16 sommets.
Il en découle que le vecteur W comporte 16 composantes d'entiers compris entre 1 et 4, où chaque entier est répété exactement 4 fois. Ainsi, la $j^{ième}$ $(j \in \{1,2,3,4\})$ apparition de l'entier $i$, $(i \in \{1,2,3,4\})$ désigne le sommet d'ordre $j$ dans le chemin $i$.

A titre d'exemple, le vecteur $W_c$=[1,4,2,1,4,3,2,1,3,4,2,1,3,2,4,3] représente le codage d'une solution du problème de coloration du graphe G décrite dans la sous-figure 3.1.c

### 3.6.2 Structure de voisinage

Chaque solution $x$ possède un voisinage $N(x) \subset X$, et chaque solution $x' \in N(x)$ est atteinte à partir de $x$ par une opération élémentaire appelée mouvement. Ce mouvement est choisi dans un voisinage modifié $N^*(x)$, où les structures de mémoire à court et à long terme sont responsables de la composition de $N^*(x)$. Plus précisément, le voisinage modifié $N^*(x)$ est le résultat du maintien d'un historique des états rencontrés lors de la recherche.

Dans les stratégies basées sur la structure de mémoire à court terme, $N^*(x)$ devient un sous-ensemble de $N(x)$, dans lequel la liste tabu a identifié les éléments de $N(x)$ à exclure de $N^*(x)$. Dans les stratégies utilisant la structure de mémoire à plus long terme, $N^*(x)$ peut également être étendu pour inclure des solutions qui ne se trouvent généralement pas dans $N(x)$.
En se basant sur ces deux stratégies, l'algorithme TS peut être considéré comme une méthode de voisinage dynamique, où le voisinage de $x$ n'est pas un ensemble statique, mais plutôt un ensemble qui peut changer en fonction de l'historique de la recherche (Glover, 1989).

Parmi les mouvements possibles susceptibles d'engendrer des solutions voisines réalisables, nous citons :

- l'inversion de deux éléments successifs dans la solution courante.
- la permutation de deux éléments distincts.
- le déplacement d'un élément de sa place d'origine à une nouvelle place.

Pour la génération du voisinage d'une solution courante au niveau de l'algorithme TS, nous avons adopté une structure de mémoire à court terme et utilisé deux types de mouvements : la permutation de deux éléments distincts et le déplacement d'un élément de sa position d'origine vers une nouvelle position. Ces mouvements s'alternent entre eux selon la stratégie développée dans l'algorithme 3.10.

**Algorithme 3.9** Génération de voisinage

| | | |
|---|---|---|
| | S | : une solution |
| **Donnée :** | Taille | : taille de voisinage |
| | Stratégie-mouvement | : (permutation) or (déplacement) |

**Sortie :** N(S) : voisinage de S
**tant que** le nombre de solutions visitées est inférieur à Taille **faire**
(1) Générer une solution non-tabu $S'$ en appliquant Stratégie-mouvement dans S
(2) $N(S) \leftarrow N(S) \cup S'$
**fin tant que**

**Algorithme 3.10** Algorithme TS pour la coloration de graphes mixtes

| | | |
|---|---|---|
| | $G = (V, E, A)$ | : Un graphe mixte |
| **Donnée :** | I | : Nombre d'itérations |
| | Max-iter | : Nombre maximum d'iterations |

**Sortie :** $S^*$ une coloration des sommets du graphe mixte G
(1) Générer une solution initiale $S_0$
(2) $S^+ \leftarrow S_0$ et $S^* \leftarrow S_0$
(3) Choisir la permutation comme type de mouvements
**tant que** Max-iter n'est pas atteint **faire**
(4) Générer $N(S^+)$ l'ensemble de voisins non tabu de $S^+$ par l'Algorithme 3.9
(5) Sélectionner $S_v$ le meilleur voisin de $N(S^+)$
(6) $S^+ \leftarrow S_v$
(7) Mettre à jour la liste tabu
**si** $(S^+) \leq (S^*)$ **alors**
(8) $S^* \leftarrow S^+$
(9) Choisir la permutation comme type de mouvements
**sinon**
**si** $S^*$ n'est pas améliorée durant I iterations **alors**
(10) Choisissez la stratégie de déplacement comme type de mouvements
**fin si**
**fin si**
**fin tant que**

## 3.7 Conclusion

Dans ce chapitre nous avons présenté les différentes méthodes proposées pour la résolution des deux problèmes de coloration de graphes mixtes, où le premier problème s'intéresse au nombre chromatique ($\chi$) et le second à la minimisation de la somme des couleurs des extrémités terminales des chemins de longueur maximale ($\sigma$).

La résolution de ces problèmes a été abordée en proposant en premier lieu une borne inférieure et une borne supérieure pour chacun des critères. Ces bornes ont été par la suite exploitées dans les programmes linéaires à variables mixtes proposés pour modéliser les deux problèmes de coloration.

Nous avons par la suite proposé plusieurs variantes de méthode branch and bound en décrivant le fonctionnement de ses trois principales composantes à savoir la stratégie de recherche, la stratégie de séparation et la stratégie d'élagage.

De ces composantes, les stratégies adoptées se résument somme suit :

- Dans la stratégie de séparation, nous avons adopté celle de la littérature modifiée par l'ajout de deux autres vecteurs permettant : (i) de limiter le nombre d'accès à l'arbre pour récupérer les informations relatives à l'ordre des sommets colorés dans les chemins, et (ii) de mémoriser les solutions partielles appartenant aux noeuds ascendants pour permettre d'évaluer une solution partielle d'un noeud courant sans avoir recours à ses ascendants.
- Dans la stratégie de recherche, nous avons adopté le parcours DFS, dans lequel nous sélectionnons le noeud possédant la meilleure borne inférieure associée. Cette stratégie permet d'explorer plus intelligemment l'arbre de solutions et d'éviter l'utilisation excessive de la mémoire.
- Dans la stratégie d'élagage, nous avons utilisé les bornes inférieures que nous avons proposées.

Quant à la résolution approchée des problèmes, nous avons opté pour une méthode basée sur l'algorithme Tabu Search utilisant une structure dynamique de voisinage.

Le chapitre suivant est consacré à la validation des différentes bornes et méthodes de résolution proposées pour les deux problèmes de coloration de graphes mixtes.

# Chapitre 4

# Expérimentations et résultats

## Sommaire

## 4.1 Introduction

Ce dernier chapitre du manuscrit est consacré à la validation des méthodes de résolution des problèmes de coloration de graphes mixtes modélisant les problèmes d'ordonnancement décrits dans le deuxième chapitre. Nous commençons par présenter l'environnement et les outils utilisés pour les différentes expérimentations réalisées. Ensuite, nous décrivons les différentes instances de problèmes utilisées pour tester et valider les méthodes et les bornes proposées. Puis de la littérature, nous donnons une description de quelques critères de classification des instances difficiles caractérisant ce type de problèmes. Le but est de montrer que la difficulté des instances ne se limite pas uniquement à leur taille (i.e. l'ordre du graphe), mais plutôt à la structure de ces instances qui est définie par une relation entre l'ordre du graphe, le nombre de cliques maximales et le nombre de chemins de longueur maximale.

Avant de valider les résultats des méthodes de résolution proposées, en particulier les méthodes basées sur la modélisation mathématique, nous nous appuyons sur des travaux de la littérature pour montrer que l'efficacité des méthodes de résolution basées sur une modélisation mathématique dépend étroitement de l'efficacité de paramétrage du solveur adopté.

Pour cela, une étude comparative entre la résolution d'un modèle mathématique par un solveur efficacement paramétré et une résolution utilisant les paramètres définis par défaut est effectuée. Les meilleurs paramètres issus de cette comparaison sont adoptés dans la configuration du CPLEX utilisé pour la résolution des modèles mathématiques que nous proposons. Les résultats expérimentaux sont décrits en deux parties, et les mêmes indicateurs de performance sont utilisés dans chacune d'elles. La première partie décrit les expérimentations utilisées pour le problème de coloration de graphes mixtes avec le critère du nombre chromatique ($\chi$), alors que la seconde décrit les expérimentations utilisées pour la coloration de graphes mixtes optimisant la somme des couleurs des extrémités terminales des chemins de longueur maximale ($\sigma$). Dans chacune des deux parties, nous validons en premier lieu les deux bornes inférieure et supérieure proposées, puis les méthodes exactes, et enfin la méthode approchée.
Pour la validation de chaque méthode, nous utilisons les deux types d'instances (les benchmarks modifiés et les instances générées de manière aléatoire).

## 4.2 Environnement et outils de développement

Les différentes expérimentations effectuées pour la validation des modèles et des méthodes proposés ont été réalisées sur un processeur Intel (R) Core (TM) i3-2100 cadencé à 3.1 Gigahertz et possédant 6 Giga Byte de RAM avec un système d'exploitation Windows 7. De plus, les méthodes branch and bound ainsi que l'algorithme TS et les bornes proposés sont codés en C++ builder Embarcadero XE4, alors que les modèles mathématiques sont résolus en utilisant le solveur CPLEX 12.5 interfacé avec Microsoft Visual C++ 10.0.

## 4.3 Description des instances

Pour évaluer la performance des bornes et des méthodes proposées, nous avons utilisé lors des expérimentations deux types d'instances de graphes mixtes : le premier type d'instances est le résultat d'une adaptation des benchmarks job shop d'OR-library, alors que le second type d'instances regroupe des graphes mixtes qui sont générés de manière aléatoire.

Nous justifions notre choix pour les benchmarks modifiés par le fait que ces derniers fournissent une base commune sur laquelle toutes les méthodes de résolution proposées pour un problème donné peuvent être testées et comparées. A partir de cette base, il sera facile de mesurer la force et la puissance d'une méthode à partir d'une déclaration telle que "la méthode *A* a donné une solution avec un coût *C* dans un temps *D*" (Jain and Meeran, 1999).

Notons que les benchmarks job shop d'OR-library sont composés de six classes d'instances décrites comme suit : trois instances (ft06, ft10, ft20) proposées par Fisher and Thompson (1963), quarante instances (la01 - la40) de Lawrence (1984), dix instances (orb01 - orb10) d'Applegate and Cook (1991), cinq instances (abz5 - abz9) d'Adams et al. (1988), vingt instances (swv01 - swv20) de Storer et al. (1992) et quatre instances (yn1 - yn4) de Yamada and Nakano (1992). Lors de cette étude, et afin d'adapter ces benchmarks aux problèmes que nous avons traités, le temps d'exécution de chaque opération est modifié à une unité. Ainsi, le nom de chaque benchmark est modifié en ajoutant (m), par exemple, orb01 est modifié et noté par orb01m.

Ensuite pour la génération des instances de graphes mixtes, nous avons adopté celle utilisée dans les travaux d'Al-Anzi et al. (2006), avec la spécificité que les chemins de longueur maximale peuvent passer plus d'une fois, ou ne pas passer, par la même clique maximale. Dans leur génération d'instances, les auteurs ont considéré deux types d'instances générées. Le premier type est composé d'instances de graphes mixtes, d'ordre inférieur à 750 sommets, avec des chemins de longueur maximale différentes, alors que le second type est composé d'instances de moins de 200 sommets, avec des chemins de même longueur maximale. Précisons que les instances du second type sont qualifiées, par les auteurs, de problèmes difficiles comparativement aux instances du premier type.
De ce fait, nous avons opté pour ce dernier type d'instances de graphes mixtes et avons utilisé les mêmes paramètres de génération adoptés par les auteurs à savoir : l'ordre des graphes mixtes est fixé à 200, le nombre de chemins de longueur maximale ($n$) est choisi dans l'ensemble {10, 20}, et le nombre de cliques maximales ($m$) est choisi dans l'ensemble {15, 17, 18} pour $n$=10 et dans l'ensemble {33, 35, 40} pour $n$=20. Notons que pour chaque combinaison de ces paramètres, dix instances de graphes mixtes sont générées.

Enfin, les caractéristiques des benchmarks modifiés (resp. des instances générées) sont détaillées dans le tableau 4.1 (resp. tableau 4.2), où on peut lire pour chaque instance, |V| le nombre de sommets du graphe mixte associé à l'instance, |A| le nombre d'arcs, ainsi que

|E| le nombre d'arêtes du benchmark ou bien le nombre moyen d'arêtes des dix instances générées, et enfin $m$ le nombre de cliques maximales et $n$ le nombre de chemins de longueur maximale.

TABLEAU 4.1 – Caractéristiques des instances de graphes mixtes issues des benchmarks modifiés

| Instance | \|V\| | \|A\| | \|E\| | m | n | Instance | \|V\| | \|A\| | \|E\| | m | n |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ft06m | 36 | 30 | 90 | 6 | 6 | la29m | 200 | 180 | 1900 | 10 | 20 |
| ft10m | 100 | 90 | 450 | 10 | 10 | la30m | 200 | 180 | 1900 | 10 | 20 |
| ft20m | 100 | 80 | 950 | 5 | 20 | la31m | 300 | 270 | 4350 | 10 | 30 |
| orb01m | 100 | 90 | 450 | 10 | 10 | la32m | 300 | 270 | 4350 | 10 | 30 |
| orb02m | 100 | 90 | 450 | 10 | 10 | la33m | 300 | 270 | 4350 | 10 | 30 |
| orb03m | 100 | 90 | 450 | 10 | 10 | la34m | 300 | 270 | 4350 | 10 | 30 |
| orb04m | 100 | 90 | 450 | 10 | 10 | la35m | 300 | 270 | 4350 | 10 | 30 |
| orb05m | 100 | 90 | 450 | 10 | 10 | la36m | 225 | 210 | 1575 | 15 | 15 |
| orb06m | 100 | 90 | 450 | 10 | 10 | la37m | 225 | 210 | 1575 | 15 | 15 |
| orb07m | 100 | 90 | 450 | 10 | 10 | la38m | 225 | 210 | 1575 | 15 | 15 |
| orb08m | 100 | 90 | 450 | 10 | 10 | la39m | 225 | 210 | 1575 | 15 | 15 |
| orb09m | 100 | 90 | 450 | 10 | 10 | la40m | 225 | 210 | 1575 | 15 | 15 |
| orb10m | 100 | 90 | 450 | 10 | 10 | swv01m | 200 | 180 | 1900 | 10 | 20 |
| la01m | 50 | 40 | 225 | 5 | 10 | swv02m | 200 | 180 | 1900 | 10 | 20 |
| la02m | 50 | 40 | 225 | 5 | 10 | swv03m | 200 | 180 | 1900 | 10 | 20 |
| la03m | 50 | 40 | 225 | 5 | 10 | swv04m | 200 | 180 | 1900 | 10 | 20 |
| la04m | 50 | 40 | 225 | 5 | 10 | swv05m | 200 | 180 | 1900 | 10 | 20 |
| la05m | 50 | 40 | 225 | 5 | 10 | swv06m | 300 | 280 | 2850 | 15 | 20 |
| la06m | 75 | 60 | 525 | 5 | 15 | swv07m | 300 | 280 | 2850 | 15 | 20 |
| la07m | 75 | 60 | 525 | 5 | 15 | swv08m | 300 | 280 | 2850 | 15 | 20 |
| la08m | 75 | 60 | 525 | 5 | 15 | swv09m | 300 | 280 | 2850 | 15 | 20 |
| la09m | 75 | 60 | 525 | 5 | 15 | swv10m | 300 | 280 | 2850 | 15 | 20 |
| la10m | 75 | 60 | 525 | 5 | 15 | swv11m | 500 | 450 | 12250 | 10 | 50 |
| la11m | 100 | 80 | 950 | 5 | 20 | swv12m | 500 | 450 | 12250 | 10 | 50 |
| la12m | 100 | 80 | 950 | 5 | 20 | swv13m | 500 | 450 | 12250 | 10 | 50 |
| la13m | 100 | 80 | 950 | 5 | 20 | swv14m | 500 | 450 | 12250 | 10 | 50 |
| la14m | 100 | 80 | 950 | 5 | 20 | swv15m | 500 | 450 | 12250 | 10 | 50 |
| la15m | 100 | 80 | 950 | 5 | 20 | swv16m | 500 | 450 | 12250 | 10 | 50 |
| la16m | 100 | 90 | 450 | 10 | 10 | swv17m | 500 | 450 | 12250 | 10 | 50 |
| la17m | 100 | 90 | 450 | 10 | 10 | swv18m | 500 | 450 | 12250 | 10 | 50 |
| la18m | 100 | 90 | 450 | 10 | 10 | swv19m | 500 | 450 | 12250 | 10 | 50 |
| la19m | 100 | 90 | 450 | 10 | 10 | swv20m | 500 | 450 | 12250 | 10 | 50 |
| la20m | 100 | 90 | 450 | 10 | 10 | abz05m | 100 | 90 | 450 | 10 | 10 |
| la21m | 150 | 135 | 1050 | 10 | 15 | abz06m | 300 | 280 | 2850 | 15 | 20 |
| la22m | 150 | 135 | 1050 | 10 | 15 | abz07m | 300 | 280 | 2850 | 15 | 20 |
| la23m | 150 | 135 | 1050 | 10 | 15 | abz08m | 300 | 280 | 2850 | 15 | 20 |
| la24m | 150 | 135 | 1050 | 10 | 15 | abz09m | 300 | 280 | 2850 | 15 | 20 |
| la25m | 150 | 135 | 1050 | 10 | 15 | yn01m | 400 | 380 | 3800 | 20 | 20 |
| la26m | 200 | 180 | 1900 | 10 | 20 | yn02m | 400 | 380 | 3800 | 20 | 20 |
| la27m | 200 | 180 | 1900 | 10 | 20 | yn03m | 400 | 380 | 3800 | 20 | 20 |
| la28m | 200 | 180 | 1900 | 10 | 20 | yn04m | 400 | 380 | 3800 | 20 | 20 |

TABLEAU 4.2 – Caractéristiques des instances de graphes mixtes générées

| Instance | \|V\| | \|A\| | \|E\| | m | n | Instance | \|V\| | \|A\| | \|E\| | m | n |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ASAA 01-10 | 200 | 190 | 1327,5 | 15 | 10 | ASAA 31-40 | 200 | 180 | 605,2 | 33 | 20 |
| ASAA 11-20 | 200 | 190 | 1164,7 | 17 | 10 | ASAA 41-50 | 200 | 180 | 570,7 | 35 | 20 |
| ASAA 21-30 | 200 | 190 | 1114,1 | 18 | 10 | ASAA 50-60 | 200 | 180 | 488,1 | 40 | 20 |

### 4.3.1 Critère de classification des instances difficiles

De manière générale, la procédure de génération des benchmarks job shop qu'on trouve dans la littérature, consiste à diviser l'ensemble des $m$ machines en $k$ sous-ensembles de taille égale ($1 \leq k \leq m$). Puis, pour un job donné, une séquence de machines est générée en passant par toutes les machines d'un même ensemble avec des affectations uniformément réparties, pour ensuite passer à un ensemble suivant de machines.

Dans cette procédure de génération de benchmarks, une classification des instances consiste à considérer comme instances faciles celles générées avec $k = 1$, et comme instances difficiles celles générées en considérant $k \geq 2$.

Dans les travaux de Jain and Meeran (1999), les auteurs apportent plus de précision sur le critère de classification des instances difficiles. En effet, ils notent le fait étrange que des instances de problèmes tels que swv 3-4 avec seulement 200 opérations et abz 7-9 avec 300 opérations soient toujours ouvertes, alors que des instances de Taillard (1993) telles que td 71-80 avec 2000 opérations ont été résolues de manière optimale dans un laps de temps relativement court après avoir été générées.

Une classification des instances difficiles est alors caractérisée en se basant sur la structure des instances de problèmes ouvertes de la littérature. Selon cette structure, la notion d'instances difficiles peut être définie comme suit :

- Une instance de N opérations peut être considérée comme difficile lorsque $N \geq 200$, avec $n \geq 15$, $m \geq 10$ et $n < 2,5m$.
- Une instance est très difficile lorsque $k = 2$ et dans cette situation $n$ ne doit pas être inférieur ou égal à $2,5m$.

## 4.4 Configuration efficace du solveur CPLEX

Les logiciels de résolution des programmes linéaires à variables mixtes sont hautement paramétrés, et trouver les meilleurs paramètres permettant de les résoudre avec une meilleure performance est un problème difficile. Ainsi, il en découle d'une part que l'efficacité des méthodes de résolution basées sur la programmation linéaire à variables mixtes est elle même dépendante de l'efficacité de paramétrage du solveur adopté. D'autre part, lorsque le nombre de paramètres est important, trouver une configuration du solveur qui conduit à de meilleures performances est considéré comme problème d'optimisation difficile (le manuel de référence des paramètres CPLEX comporte 221 pages décrivant 135 paramètres pouvant affecter le processus de recherche d'une solution optimale).

On peut toutefois trouver plus de précisions dans le travail de Hutter et al. (2010), où les auteurs ont identifié 76 paramètres pouvant être fixés soigneusement pour optimiser les performances de résolution d'un programme linéaire à variables mixtes. Ces 76 paramètres

comportent 12 paramètres de prétraitement, 17 paramètres de stratégie MIP, 11 paramètres déterminant le type de coupes à ajouter et la façon de les utiliser en terme d'agressivité, 9 paramètres numériques MIP, 10 paramètres relatifs au simplex, 6 paramètres d'optimisation de barrière, et 11 autres paramètres. Exploitant le fait que 4 de ces paramètres sont conditionnels à d'autres prenants certaines valeurs, ces 76 paramètres ont donné lieu à $1,9 \times 10^{47}$ configurations distinctes.

Dans l'étude menée par Hutter et al. (2010), un algorithme est proposé pour déterminer les meilleurs paramètres d'une configuration efficace en utilisant trois solveurs : CPLEX, GUROBI et LPSOLVE. En vue de cela, ils ont résolu plusieurs instances de programmes linéaires sur chacun de ces solveurs. Au niveau de chaque résolution, ils ont déroulé leur algorithme pour trouver les meilleurs paramètres de configuration du solveur utilisé. Les résultats de leur étude sont jugés très efficaces comparativement aux résultats obtenus de la résolution par CPLEX utilisant la configuration par défaut.

Au vu de ce constat, nous nous sommes proposés de configurer CPLEX, en utilisant les 76 paramètres issus des travaux de Hutter et al. (2010), et de faire ensuite une étude comparative entre le CPLEX configuré par défaut et le CPLEX efficacement configuré par ces paramètres (cf. tableau 4.3).

Dans l'étude comparative, nous avons résolu le modèle mathématique proposé pour la coloration de graphes mixtes avec le critère $\chi$ (MILP$_\chi$), et nous utilisons les dix instances de graphes mixtes issues des benchmarks la31m - la40m (cf. tableau 4.3).

TABLEAU 4.3 – Efficacité des paramètres adoptés pour la configuration du CPLEX

| Instance | Cplex configuré par défaut | | | | Cplex configuré par les 76 paramètres | | | |
|---|---|---|---|---|---|---|---|---|
| | $\chi$ | noeuds | iterations | CPU | $\chi$ | noeuds | iterations | CPU |
| la31m | 32 | 645904 | 103789909 | 4194 | 32 | 3726 | 919837 | 75 |
| la32m | 34 | 331912 | 145633280 | 5400 | 31 | 49632 | 7778554 | 183 |
| la33m | 34 | 766847 | 96221258 | 5400 | 30 | 494445 | 119090054 | 3696 |
| la34m | 33 | 288052 | 130703445 | 5400 | 31 | 101846 | 16369269 | 694 |
| la35m | 35 | 338191 | 81820759 | 5400 | 31 | 689645 | 133686798 | 5400 |
| la36m | 22 | 41821 | 53738798 | 2556 | 22 | 250756 | 22647902 | 480 |
| la37m | 27 | 182142 | 84701872 | 5400 | 23 | 4802 | 1812290 | 110 |
| la38m | 23 | 199492 | 163098922 | 5400 | 22 | 2028038 | 233281325 | 4900 |
| la39m | 22 | 45705 | 65664712 | 3300 | 22 | 377402 | 67626750 | 1450 |
| la40m | 28 | 206609 | 95684936 | 5400 | 24 | 224854 | 35248109 | 1360 |

D'après les résultats présentés dans le tableau 4.3, nous constatons que la Résolution du Modèle par le Solveur utilisant la Configuration Paramétrée (RMSCP) est nettement meilleure que la Résolution du Modèle par le Solveur utilisant la Configuration par Default (RMSCD). En effet, parmi les 10 instances testées, nous constatons que 9 instances sont résolues de manière optimale par RMSCP, alors que seulement 3 instances sont résolues de manière optimale par RMSCD. De plus, le temps de résolution est nettement plus réduit si on compare les instances résolues de manière optimale dans les deux cas. Ainsi, les paramètres validés

dans cette étude comparative [1] sont adoptés dans la configuration du solveur CPLEX utilisé pour la résolution des modèles mathématiques proposés dans le chapitre précédent, et dont les résultats sont donnés ci-après.

## 4.5 Indicateurs de performance des méthodes de résolution

Lors des expérimentations réalisées sur les différents types d'instances de problèmes traitées pour la résolution des méthodes exactes, nous avons limité le temps de calcul de la recherche d'une solution optimale à 5400 secondes. De plus et afin de mener une analyse sur la performance et l'efficacité des différentes méthodes proposées, nous présentons ci-après quelques indicateurs de performance :

- $\mathrm{RD}_{opt}$ : pourcentage de déviation entre la solution obtenue et la solution optimale $\frac{(s-s_{opt})\times 100}{s_{opt}}$, où s est égale à $\chi$ (resp. $\sigma$) pour la coloration de graphes mixtes avec le critère $\chi$ (resp. $\sigma$)
- $\mathrm{RD}_{\mathrm{LB}}$ : pourcentage de déviation entre la solution obtenue et la borne inférieure $\frac{(s-\mathrm{LB})\times 100}{\mathrm{LB}}$, où s est égale à $\chi$ (resp. $\sigma$) et LB est égale à $\mathrm{LB}_\chi$ (resp. $\mathrm{LB}_\sigma$) pour la coloration de graphes mixtes avec le critère $\chi$ (resp. $\sigma$)
- $\mathrm{N}_{opt}$ : nombre d'instances résolues de manière optimale parmi l'ensemble des instances testées
- Noeuds : nombre de noeuds visités dans l'arbre de solutions utilisé par branch and bound
- CPU : temps de résolution

## 4.6 Expérimentations pour la coloration de graphes mixtes avec le critère $\chi$

Dans cette section nous nous intéressons, en premier lieu, à la validation des deux bornes inférieure et supérieure proposées pour le nombre chromatique $\chi$. Ensuite, nous présentons une étude expérimentale pour l'évaluation de la performance des méthodes exactes et de la méthode approchée proposées pour la résolution du problème de coloration de graphes mixtes avec le critère du nombre chromatique $\chi$.

### 4.6.1 Validation des bornes proposées pour $\chi$

Cette partie s'intéresse à l'évaluation de la borne inférieure $\mathrm{LB}_\chi$ proposée dans le chapitre précédent pour le nombre chromatique (cf. chap3, § 3.2.1). Pour juger de la qualité de

1. http ://www.cs.ubc.ca/labs/beta/Projects/MIP-Config/.

cette borne, une étude comparative a été réalisée avec la borne inférieure $LB_{\chi 1}$ de Sotskov et al. (2001). Ainsi, des expérimentations ont été menées en s'appuyant sur les benchmarks modifiés, dont les résultats sont synthétisés dans le tableau 4.4. A partir de ce tableau, nous pouvons lire respectivement pour chaque instance, les valeurs de $LB_{\chi 1}$, $LB_{\chi}$ ainsi que $UB_{\chi 0}$ et $UB_{\chi}$. Ces deux dernières représentent respectivement la borne supérieure de Sotskov et al. (2001) et la borne supérieure que nous avons proposée dans la section 3.3 du chapitre 3.

TABLEAU 4.4 – Bornes inférieures et supérieures pour $\chi$ (benchmarks modifiés)

| Instance | $LB_{\chi 1}$ | $LB_{\chi}$ | $UB_{\chi 0}$ | $UB_{\chi}$ | Instance | $LB_{\chi 1}$ | $LB_{\chi}$ | $UB_{\chi 0}$ | $UB_{\chi}$ |
|---|---|---|---|---|---|---|---|---|---|
| ft06m | 8 | 8 | 36 | 10 | la29m | 21 | 21 | 200 | 29 |
| ft10m | 16 | 16 | 100 | 22 | la30m | 21 | 21 | 200 | 27 |
| ft20m | 22 | 22 | 100 | 30 | la31m | 32 | 32* | 300 | 34 |
| orb01m | 16 | 16 | 100 | 23 | la32m | 31 | 31* | 300 | 37 |
| orb02m | 14 | 16 | 100 | 20 | la33m | 30 | 30* | 300 | 37 |
| orb03m | 17 | 17 | 100 | 26 | la34m | 31 | 31* | 300 | 37 |
| orb04m | 17 | 17 | 100 | 19 | la35m | 30 | 30 | 300 | 37 |
| orb05m | 16 | 16 | 100 | 21 | la36m | 18 | 18 | 225 | 25 |
| orb06m | 16 | 16 | 100 | 23 | la37m | 19 | 22 | 225 | 28 |
| orb07m | 14 | 16 | 100 | 20 | la38m | 19 | 20 | 225 | 26 |
| orb08m | 17 | 17 | 100 | 26 | la39m | 19 | 20 | 225 | 26 |
| orb09m | 17 | 17 | 100 | 19 | la40m | 18 | 20 | 225 | 29 |
| orb10m | 16 | 16 | 100 | 21 | swv01m | 26 | 26 | 200 | 39 |
| la01m | 10 | 10* | 50 | 12 | swv02m | 25 | 25 | 200 | 38 |
| la02m | 11 | 11* | 50 | 13 | swv03m | 26 | 26 | 200 | 38 |
| la03m | 12 | 12* | 50 | 13 | swv04m | 25 | 25 | 200 | 38 |
| la04m | 11 | 11* | 50 | 12 | swv05m | 25 | 25 | 200 | 38 |
| la05m | 10 | 10 | 50 | 12 | swv06m | 28 | 28 | 200 | 44 |
| la06m | 15 | 15* | 75 | 17 | swv07m | 28 | 28 | 300 | 41 |
| la07m | 16 | 16* | 75 | 19 | swv08m | 28 | 28 | 300 | 44 |
| la08m | 15 | 15* | 75 | 18 | swv09m | 29 | 29 | 300 | 44 |
| la09m | 15 | 15* | 75 | 16 | swv10m | 29 | 29 | 300 | 41 |
| la10m | 16 | 16* | 75 | 17 | swv11m | 55 | 55 | 500 | 92 |
| la11m | 21 | 21* | 100 | 24 | swv12m | 55 | 55 | 500 | 90 |
| la12m | 20 | 20* | 100 | 20 | swv13m | 55 | 55 | 500 | 88 |
| la13m | 20 | 20* | 100 | 24 | swv14m | 55 | 55 | 500 | 90 |
| la14m | 20 | 20* | 100 | 20 | swv15m | 55 | 55 | 500 | 91 |
| la15m | 20 | 20* | 100 | 24 | swv16m* | 51 | 51 | 500 | 52 |
| la16m | 13 | 13 | 100 | 17 | swv17m* | 50 | 50 | 500 | 54 |
| la17m | 15 | 15* | 100 | 17 | swv18m* | 50 | 50 | 500 | 55 |
| la18m | 13 | 14 | 100 | 17 | swv19m* | 50 | 50 | 500 | 55 |
| la19m | 14 | 15* | 100 | 18 | swv20m* | 50 | 50 | 500 | 54 |
| la20m | 12 | 14 | 100 | 18 | abz05m | 12 | 13 | 100 | 17 |
| la21m | 17 | 18 | 150 | 22 | abz06m | 24 | 24 | 300 | 29 |
| la22m | 17 | 18 | 150 | 22 | abz07m | 24 | 24 | 300 | 29 |
| la23m | 16 | 17 | 150 | 21 | abz08m | 24 | 24 | 300 | 30 |
| la24m | 17 | 17* | 150 | 20 | abz09m | 23 | 24 | 300 | 33 |
| la25m | 17 | 18 | 150 | 22 | yn01m | 23 | 24 | 400 | 33 |
| la26m | 22 | 22 | 200 | 24 | yn02m | 23 | 25 | 400 | 34 |
| la27m | 22 | 22* | 200 | 29 | yn03m | 24 | 27 | 400 | 38 |
| la28m | 22 | 22* | 200 | 24 | yn04m | 25 | 28 | 400 | 40 |

* $LB_{\chi}$ coïncide avec l'optimum

Nous pouvons remarquer que $LB_\chi$ est plus efficace que $LB_{\chi 1}$. En effet, parmi les 82 benchmarks testés, les valeurs de $LB_\chi$ sont remarquablement meilleures que celles de $LB_{\chi 1}$ pour 19 benchmarks modifiés, et sont égales à $LB_{\chi 1}$ pour le reste des benchmarks.

Notons aussi, que comparativement aux valeurs optimales des solutions obtenues par des méthodes de résolution exacte pour les différentes instances, la valeur de $LB_{\chi 1}$ coïncide avec la solution optimale de 29 benchmarks, ce qui atteste qu'elle est suffisamment serrée (cf. tableaux 4.4, 4.6 et 4.9).

Considérons maintenant l'ensemble des instances générées, où les résultats sont présentés dans le tableau 4.5. Les colonnes libellées (a), (b) et (c) de ce tableau décrivent respectivement le nombre de fois en pourcentage où $LB_\chi > LB_{\chi 1}$, $LB_\chi = LB_{\chi 1}$ et $LB_\chi$ coïncide avec la solution optimale. De ces résultats, nous constatons que sur les 60 instances, $LB_\chi$ est strictement supérieure à $LB_{\chi 1}$ pour 53 instances, et elles sont égales uniquement pour 7 instances. De plus, $LB_\chi$ coïncide avec la solution optimale pour 27 instances.

TABLEAU 4.5 – Bornes inférieures pour $\chi$ (instances générées)

| Instances | a | b | c | Instances | a | b | c |
|---|---|---|---|---|---|---|---|
| ASAA 01-10 | 100 | 0 | 0 | ASAA 31-40 | 90 | 10 | 80 |
| ASAA 11-20 | 100 | 0 | 0 | ASAA 41-50 | 70 | 30 | 80 |
| ASAA 21-30 | 100 | 0 | 30 | ASAA 51-60 | 70 | 30 | 80 |

a : Nombre de fois en pourcentage où $LB_\chi > LB_{\chi 1}$
b : Nombre de fois en pourcentage où $LB_\chi = Lb_{\chi 1}$
c : Nombre de fois en pourcentage où $LB_\chi$ coïncide avec la solution optimale

Concentrons-nous maintenant sur la borne supérieure $UB_\chi$ proposée dans la section 3.3 du chapitre 3. Nous remarquons qu'il est bien clair que cette borne est largement meilleure que la borne supérieure $UB_{\chi 0}$ de Sotskov et al. (2001). Cette dernière est triviale, puisque sa valeur correspond au nombre des sommets du graphe mixte considéré.

A titre illustratif, considérons l'ensemble des benchmarks modifiés. Dans le tableau 4.4, les valeurs d'$UB_{\chi 0}$ sont détaillées dans les colonnes 4 et 9, alors que les valeurs d'$UB_\chi$ sont détaillées dans les colonnes 5 et 10. De ces valeurs, une comparaison entre ces deux bornes est bien évidente en faveur d'$UB_\chi$.

## 4.6.2 Résultats des méthodes exactes

Dans cette partie, nous présentons une étude comparative des méthodes exactes proposées selon les expérimentations réalisées, d'une part sur les instances issues des benchmarks modifiés, et d'autre part sur celles générées de manière aléatoire (cf. chap4, §4.3).

### 4.6.2.1 Benchmarks modifiés

Le tableau 4.6 synthétise les résultats de la résolution des benchmarks modifiés par les deux variantes de l'algorithme branch and bound ($BB1_\chi$ et $BB2_\chi$), ainsi que ceux du modèle

TABLEAU 4.6 – Résolution exacte des benchmarks modifiés : (cas de $\chi$)

| Instance | $BB1_\chi$ | | | $BB2_\chi$ | | | $MILP_\chi$ | |
|---|---|---|---|---|---|---|---|---|
| | $\chi$ | Noeuds | CPU | $\chi$ | Noeuds | CPU | $\chi$ | CPU |
| ft06m | 9 | 91 | 1 | 9 | 38 | 0 | 9 | 1 |
| ft10m | 18 | 579 249 122 | 5400 | 18 | 12 271 401 | 160 | 18 | 853 |
| ft20m | 23 | 551 387 347 | 5400 | 23 | 386 284 658 | 5400 | 23 | 5118 |
| orb01m | 18 | 200 430 309 | 1848 | 18 | 2 533 722 | 26 | 18 | 139 |
| orb02m | 17 | 4 111 036 | 39 | 17 | 212 757 | 2 | 17 | 70 |
| orb03m | 18 | 22 819 383 | 195 | 18 | 913 584 | 10 | 18 | 156 |
| orb04m | 18 | 85 593 222 | 819 | 18 | 140 238 | 1 | 18 | 1 |
| orb05m | 17 | 88 329 | 0 | 17 | 12 502 | 1 | 17 | 8 |
| orb06m | 18 | 200 430 309 | 1836 | 18 | 2 533 722 | 27 | 18 | 139 |
| orb07m | 17 | 4 111 036 | 39 | 17 | 212 757 | 2 | 17 | 70 |
| orb08m | 18 | 22 819 383 | 192 | 18 | 913 584 | 10 | 18 | 157 |
| orb09m | 18 | 85 593 222 | 814 | 18 | 140 238 | 1 | 18 | 1 |
| orb10m | 17 | 88 329 | 1 | 17 | 12 502 | 1 | 17 | 9 |
| la01m | 10 | 3 246 | 1 | 10 | 265 | 1 | 10 | 1 |
| la02m | 11 | 686 | 1 | 11 | 257 | 1 | 11 | 1 |
| la03m | 12 | 545 103 | 2 | 12 | 205 958 | 1 | 12 | 1 |
| la04m | 11 | 363 | 1 | 11 | 202 | 1 | 11 | 1 |
| la05m | 11 | 1 041 053 | 4 | 11 | 443 | 1 | 11 | 1 |
| la06m | 15 | 1 163 | 1 | 15 | 1 121 | 1 | 15 | 1 |
| la07m | 16 | 1 219 | 1 | 16 | 1 018 | 1 | 16 | 1 |
| la08m | 15 | 1 832 | 1 | 15 | 957 | 1 | 15 | 1 |
| la09m | 15 | 589 425 | 3 | 15 | 58 281 | 1 | 15 | 1 |
| la10m | 16 | 2 587 | 1 | 16 | 1 569 | 1 | 16 | 1 |
| la11m | 21 | 4 890 | 1 | 21 | 4 236 | 1 | 21 | 2 |
| la12m | 20 | 9 164 | 1 | 20 | 1 | 1 | 20 | 2 |
| la13m | 20 | 17 288 | 1 | 20 | 6 051 | 1 | 20 | 3 |
| la14m | 20 | 8 277 | 1 | 20 | 1 | 1 | 20 | 3 |
| la15m | 20 | 124 481 | 1 | 20 | 10 315 | 1 | 20 | 4 |
| la16m | 15 | 23 080 977 | 189 | 15 | 886 779 | 8 | 15 | 1 |
| la17m | 15 | 7 926 | 1 | 15 | 1 543 | 1 | 15 | 1 |
| la18m | 15 | 412 182 | 2 | 15 | 24 406 | 1 | 15 | 4 |
| la19m | 15 | 34 210 | 1 | 15 | 1 024 | 1 | 15 | 1 |
| la20m | 15 | 11 043 139 | 77 | 15 | 293 | 1 | 15 | 1 |
| la21m | 19 | 409 924 088 | 5400 | 19 | 5 057 070 | 62 | 19 | 679 |
| la22m | 19 | 475 559 667 | 5400 | 18 | 36 900 | 1 | 18 | 23 |
| la23m | 18 | 453 328 402 | 5400 | 18 | 310 750 745 | 4121 | 18 | 3994 |
| la24m | 17 | 710 379 011 | 5400 | 17 | 142766 | 1 | 17 | 4 |
| la25m | 19 | 523 224 852 | 5400 | 18 | 37 371 962 | 388 | 18 | 138 |
| la26m | 23 | 293 348 115 | 5400 | 23 | 348 612 653 | 5400 | 22 | 21 |
| la27m | 24 | 199 115 044 | 5400 | 22 | 124 963 396 | 2076 | 22 | 226 |
| la28m | 24 | 371 175 166 | 5400 | 22 | 983 235 | 16 | 22 | 7 |
| la29m | 24 | 153 064 605 | 5400 | 23 | 143 257 185 | 5400 | 23 | 5400 |
| la30m | 26 | 247 816 432 | 5400 | 23 | 299 390 131 | 5400 | 23 | 5400 |
| la31m | 35 | 30 620 084 | 5400 | 33 | 32 691 249 | 5400 | 32 | 75 |
| la32m | 35 | 10 584 462 | 5400 | 32 | 25 951 941 | 5400 | 31 | 183 |
| la33m | 36 | 12 909 158 | 5400 | 33 | 22 000 000 | 5400 | 30 | 3696 |
| la34m | 35 | 4 685 873 | 5400 | 32 | 31 509 847 | 5400 | 31 | 694 |
| la35m | 37 | 21 225 657 | 5400 | 33 | 39 480 899 | 5400 | 31 | 5400 |
| la36m | 24 | 347 356 747 | 5400 | 23 | 351 000 000 | 5400 | 22 | 480 |
| la37m | 26 | 392 056 161 | 5400 | 24 | 304 605 141 | 5400 | 23 | 110 |
| la38m | 25 | 403 327 787 | 5400 | 23 | 322 283 814 | 5400 | 22 | 4900 |
| la39m | 25 | 440 103 493 | 5400 | 23 | 319 172 516 | 5400 | 22 | 1450 |
| la40m | 27 | 409 469 207 | 5400 | 25 | 386 570 477 | 5400 | 24 | 1360 |

mathématique ($MILP_\chi$). Rappelons que dans la stratégie d'élagage, $BB1_\chi$ utilise la borne inférieure $LB_{\chi 1}$ de Sotskov et al. (2001), et $BB2_\chi$ utilise la borne inférieure $LB_\chi$ que nous avons proposée. Ainsi dans ce tableau, les résultats sont présentés comme suit : les colonnes 2, 5 et 8 représentent le nombre chromatique ($\chi$) obtenu respectivement par $BB1_\chi$, $BB2_\chi$ et $MILP_\chi$, alors que les colonnes 4, 7 et 9 représentent le temps de résolution (CPU) correspondant respectivement aux $BB1_\chi$, $BB2_\chi$ et $MILP_\chi$. Enfin, les colonnes 3 et 6 rapportent le nombre de noeuds (Noeuds) explorés dans l'arbre de solutions par respectivement $BB1_\chi$ et $BB2_\chi$.

Notre objectif est de faire une analyse sur la performance de ces méthodes, en s'appuyant principalement sur les différents indicateurs de performance donnés dans la section 4.5.

D'après les résultats, nous pouvons constater que le modèle ($MILP_\chi$) a résolu des instances de graphes mixtes d'ordre inférieur ou égal à 300 sommets. En effet, il a trouvé 50 solutions optimales pour les 53 benchmarks testés dans un délai moyen de résolution plutôt court (soit moins de 484 secondes). Cependant, l'optimum n'est pas atteint dans le délai imposé pour seulement trois instances (la29m, la30m et la35m), et le pourcentage de déviation par rapport à la borne inférieure $RD_{LB}$ pour ces instances est respectivement (9,52%, 9,52% et 3,33%).

Nous remarquons aussi que les solutions obtenues par la résolution du modèle ($MILP_\chi$) sont de meilleure ou de même qualité que celles fournies par les méthodes $BB1_\chi$ ou $BB2_\chi$. Néanmoins, le temps de résolution (CPU) des benchmarks résolus à la fois par $MILP_\chi$ et $BB2_\chi$ est généralement meilleur pour $BB1_\chi$.

Pour ce qui est de résultats des expérimentations obtenus par les variantes de l'algorithme branch and bound $BB1_\chi$ et $BB2_\chi$, nous constatons que $BB1_\chi$ et $BB2_\chi$ ont aussi résolu des benchmarks d'ordre inférieur ou égal à 300 sommets. Toutefois l'optimum n'est pas atteint, dans le délai imposé, pour les benchmarks d'ordre supérieur à 200 sommets.

Dans un autre point de vue, si nous comparons les deux méthodes $BB1_\chi$ et $BB2_\chi$ indépendamment des résultats obtenus par le $MILP_\chi$, nous remarquons que $BB2_\chi$ est généralement plus performante que $BB1_\chi$ de point de vue de :

- nombre de solutions optimales obtenues ($N_{opt}$) : parmi les 53 benchmarks testés, $BB2_\chi$ retourne 39 solutions optimales, alors que $BB1_\chi$ ne retourne que 31 solutions optimales. Dans cette comparaison, nous n'avons pas considéré les solutions optimales déduites de la comparaison avec $MILP_\chi$, puisque ces dernières sont obtenues sans que les noeuds de l'arbre de solutions soient totalement stérilisés dans la limite du temps imposé. Il s'agit précisément des benchmarks ft10m, ft20m, la21m, la23m, la24m et la26m résolus par $BB1_\chi$, et seulement ft20m résolu par $BB2_\chi$.
- temps de résolution optimale des benchmarks (CPU) : ce temps est en faveur de $BB2_\chi$ pour tout benchmark résolu de manière optimale à la fois par les deux méthodes $BB1_\chi$ et $BB2_\chi$.
- nombre de noeuds explorés dans l'arbre de solutions (Noeuds) : pour les benchmarks

résolus de manière optimale à la fois par les deux méthodes, ce nombre est dans la plus part des cas en faveur de $BB2_{\chi}$.

– pourcentage de déviation par rapport à la solution optimale ($RD_{opt}$) : ce pourcentage est calculé pour les benchmarks résolus de manière non optimale par au moins une des deux méthodes, tout en supposant que leur solution optimale est connue. Plus précisément, il est calculé pour 14 benchmarks, et sa valeur n'excède pas les 10% (resp. 20%) pour $BB2_{\chi}$ (resp. $BB1_{\chi}$ ). De plus, tous les pourcentages liés à $BB2_{\chi}$ sont nettement inférieurs à ceux de $BB1_{\chi}$ (cf. figure 4.1).

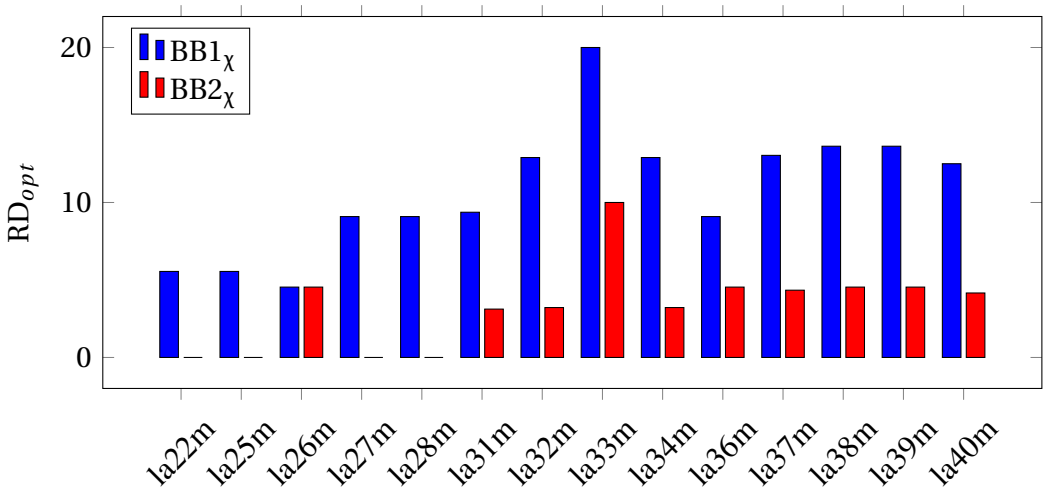


FIGURE 4.1 – Déviation par rapport à l'optimum pour les benchmarks modifiés (cas de $\chi$)

– pourcentage de déviation par rapport à la borne inférieure ($RD_{LB}$) : ce pourcentage est calculé pour les benchmarks non résolus de manière optimale par les 3 méthodes exactes. Il s'agit donc de 03 instances et leur valeur ne dépasse pas 10% pour $BB2_{\chi}$, et est comprise entre 14,25% et 23,33% pour $BB1_{\chi}$ (cf. figure 4.2).

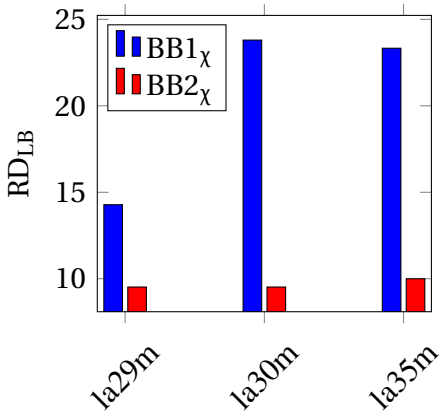


FIGURE 4.2 – Déviation par rapport à la borne inférieure pour les benchmarks modifiés (cas de $\chi$)

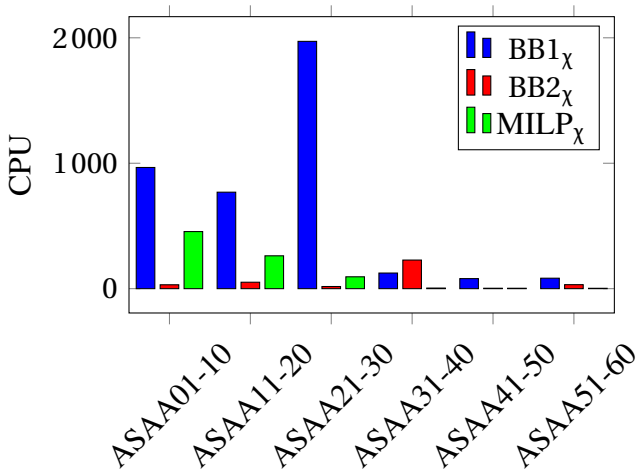


FIGURE 4.3 – Temps moyen de résolution des instances générées (cas de $\chi$)

#### 4.6.2.2 Instances générées

Cette sous-section est consacrée à l'analyse des résultats des méthodes exactes ($MILP_\chi$, $BB1_\chi$ et $BB2_\chi$) validées sur les instances de graphes mixtes générées de manière aléatoire. Ces résultats sont présentés dans le tableau 4.7, où les colonnes 2, 4 et 6 représentent le pourcentage d'instances résolues de manière optimale par respectivement $BB1_\chi$, $BB2_\chi$ et $MILP_\chi$ alors que les colonnes 3, 5 et 7 représentent le temps moyen de résolution ($\overline{CPU}$) correspondant respectivement aux $BB1_\chi$, $BB2_\chi$ et $MILP_\chi$ .

Nous commençons par analyser les résultats issus de la méthode basée sur $MILP_\chi$. Nous constatons que toutes les solutions retournées par $MILP_\chi$ sont optimales, et que le temps moyen de résolution est égale à 455,2 secondes pour le premier type d'instances, puis il diminue d'un type d'instances à un autre jusqu'à atteindre les 3 secondes pour le dernier type d'instances (cf. figure 4.3).

TABLEAU 4.7 – Résolution exacte des instances générées (cas de $\chi$)

| Instances | $BB1_\chi$ | | $BB2_\chi$ | | $MILP_\chi$ | |
|---|---|---|---|---|---|---|
| | d | $\overline{CPU}$ | d | $\overline{CPU}$ | d | $\overline{CPU}$ |
| ASAA 01-10 | 10 | 966,2 | 90 | 31,1 | 100 | 455,2 |
| ASAA 11-20 | 30 | 769,3 | 100 | 51,4 | 100 | 262 |
| ASAA 21-30 | 30 | 1971,7 | 90 | 17,2 | 100 | 94,7 |
| ASAA 31-40 | 80 | 124,6 | 100 | 228,1 | 100 | 4,8 |
| ASAA 41-50 | 70 | 80,2 | 100 | 3,4 | 100 | 3,3 |
| ASAA 51-60 | 90 | 83,5 | 100 | 32 | 100 | 3,0 |

d : Nombre de fois en pourcentage où $\chi$ est optimal

En ce qui concerne les autres résulats des expérimentations issus des deux méthodes branch and bound, nous pouvons constater en premier lieu que la méthode de résolution basée sur ($MILP_\chi$) demeure plus performante que ces deux méthodes. En second lieu, nous constatons que la méthode $BB2_\chi$ est plus performante que $BB1_\chi$. Effectivement, $BB2_\chi$ trouve 58 solutions optimales ($N_{opt}$) sur les 60 instances testées, alors que $BB1_\chi$ ne trouve que

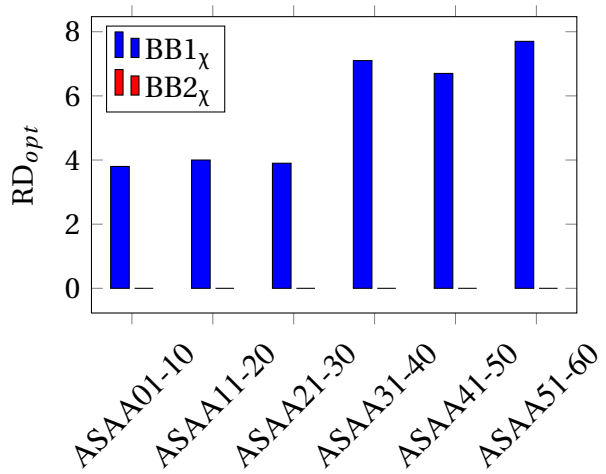


FIGURE 4.4 – Déviation moyenne par rapport à l'optimum pour les instances générées (cas de $\chi$)

seulement 31 solutions optimales.

De plus, le temps moyen de résolution optimale (CPU) est généralement en faveur de BB2$_\chi$ (cf. figure 4.3).

Nous constatons aussi la bonne qualité des solutions retournées en 5400 secondes, sans que tous les noeuds de l'arbre de solutions soient stérilisés. En effet, 12 solutions sur 29 (resp. 2 solutions sur 2) obtenues dans la limite du temps de résolution par BB1$_\chi$ (resp. par BB2$_\chi$) sont optimales suite à une comparaison avec les résultats du modèle MILP$_\chi$.

En outre, le pourcentage de déviation par rapport à la solution optimale concernant les instances résolues de manière non optimale par BB1$_\chi$ n'excède pas 7,7%. Cependant, la moyenne de ces pourcentages pour chaque type d'instances est bien illustrée dans la figure 4.4.

#### 4.6.2.3 Performances des méthodes exactes après la relaxation du temps de résolution imposé

Il est bien claire que, pour les instances de grande taille, les solveurs de programmes linéaires à variables mixtes peuvent avoir recours au swapping entre la mémoire principale et le disque pour gérer l'insuffisance de mémoire et maintenir le processus de résolution actif. Dans le cas inverse, le processus de résolution est interrompu par un message d'erreur "*Out of memory*". Sans trop entrer dans les détails du swapping, il faut juste noter que ce dernier est constitué de temps de transfert qui est directement proportionnel à la quantité de mémoire transférée. Par conséquent, ce temps qui s'ajoute à celui de la résolution peut influencer négativement sur la performance de résolution.

Notons qu'il en est de même pour la plupart des méthodes branch and bound de la littérature qui requièrent un espace mémoire proportionnel à la taille de l'arbre d'enumeration. Elles

peuvent de ce fait subir une limitation de la mémoire principale et avoir recours au swapping.

A lumière de ce qui est dit, nous montrons que les variantes de l'algorithme branch and bound proposées ($BB1_\chi$ et $BB2_\chi$) peuvent fonctionner sans subir les contraintes liées à la limitation de la mémoire principale. Ceci s'explique par le fait que (i) tous les noeuds non pendants de l'arbre sont supprimés de la mémoire, et (ii) qu'un algorithme de parcours en profondeur d'arbre (DFS) utilisant les bornes inférieures est adopté. Il en découle que le nombre de noeuds non stérilisés présents dans la mémoire principale est toujours limité (cf. chap3, § 3.5.4).

A titre illustratif, considérons les benchmarks non résolus de manière optimale par la méthode exacte $MILP_\chi$. Nous nous proposons d'étendre le temps de résolution à 86400 secondes (24 heures), et de résoudre de nouveau ces benchmarks par les méthodes $BB2_\chi$ et $MILP_\chi$.

Les résultats des nouvelles expérimentations sont présentés dans le tableau 4.8. Nous pouvons constater que la méthode $BB2_\chi$ a réussi à trouver une solution optimale pour deux des trois benchmarks, alors que la résolution du modèle $MILP_\chi$ par CPLEX n'a pas réussi à les résoudre de manière optimale en moins de 86400 secondes.

TABLEAU 4.8 – Résolution exacte de la29m, la30m et la35m sous moins de 86400 secondes : (cas de $\chi$)

| Instance | $BB2_\chi$ | | | $MILP_\chi$ | | |
|---|---|---|---|---|---|---|
| | $\chi$ | Noeuds | CPU | $\chi$ | Noeuds | CPU |
| la29m | 22 | 188 404 123 | 10560 | 23 | 1 303 191 | 86400 |
| la30m | 22 | 562 433 553 | 9183 | 23 | 1 303 191 | 86400 |
| la35m | 32 | 212 256 570 | 86400 | 31 | 39 480 899 | 86400 |

Un autre résultat que nous estimons très efficace réside dans la gestion optimisée de la mémoire principale. En effet, la suppression des noeuds non pendants de l'arbre de solutions a permis de faire fonctionner cette dernière sans avoir des soucis sur la capacité de la mémoire principale. A titre d'exemples, l'arbre de solutions de la méthode $BB2_\chi$ a généré 562 433 553 noeuds pour résoudre le benchmark la30m (cf. tableau 4.8). Comme chaque noeud de l'arbre contient au moins 30 entiers (cf. le codage des vecteurs associés aux noeuds décrit dans la § 3.5.4.1 du chapitre 3), et si nous supposons qu'aucun noeud de l'arbre n'a été supprimé, alors l'algorithme nécessiterait plus de 62 gigabytes de mémoire principale pour gérer cet arbre.

### 4.6.3 Résultats de l'algorithme TS

Cette partie est consacrée aux résultats des expérimentions de l'algorithme TS et à l'analyse de la performance de la méthode. Commençons par rappeler que l'efficacité d'un algorithme TS dépend étroitement de l'ajustement de ses paramètres. Ces derniers peuvent

être fixés selon les résultats des expérimentations préliminaires en retenant les valeurs pour lesquelles un compromis entre la qualité d'une solution et le temps de résolution est réalisé. Dans le cas du problème étudié, des expérimentations préliminaires ont été effectuées sur un ensemble d'instances et ont permis de fixer les paramètres suivants :

- nombre maximal d'itérations $Max - iter = 1000$
- taille de la liste tabou $list = n \times m$
- taille du voisinage $Vs = 1000$
- nombre d'itérations utilisées dans la structure de voisinage $l = 100$

Dans ce qui suit, nous nous sommes proposé de valider l'algorithme TS sur les deux types d'instances (benchmarks modifiés et instances générées), et comme l'algorithme TS est non déterministe, l'exécution de chacune des instances du problème est lancée exactement 5 fois.

#### 4.6.3.1 Benchmarks modifiés

Les différents résultats des expérimentations sont présentés dans le tableau 4.9, nous pouvons constater que l'algorithme TS retourne de très bonnes solutions pour la plupart des 82 benchmarks modifiés. Notons que sa performance est évaluée en considérant trois indicateurs : la meilleure solution trouvée des cinq lancements d'une même instance ($\chi_{best}$), la moyenne des 5 solutions trouvées ($\bar{\chi}$) et le maximum de leur temps de résolution (CPU).

En analysant ces résultats, nous remarquons que l'algorithme TS retourne des solutions optimales pour 49 benchmarks. L'optimalité de 20 d'entre elles n'est déduite que par comparaison avec la résolution du $MILP_{\chi}$, alors que l'optimalité de 5 d'entre elles est déduite uniquement par comparaison avec la valeur de la borne inférieure ($LB_{\chi}$).

Pour les solutions non optimales (i.e. $\chi_{best}$ est non optimale), l'indicateur sur la moyenne de $RD_{opt}$ (resp. $RD_{LB}$) est évaluée pour $\chi_{best}$ lorsque la solution optimale est connue (resp. inconnue).
A partir de ces deux indicateurs (la moyenne de $RD_{opt}$ = 4,64 et la moyenne de $RD_{LB}$ = 11,99), nous pouvons déduire que la qualité des solutions approchées est généralement très bonne. De plus, nous pouvons aussi attester que l'algorithme TS est robuste dans le sens où la moyenne de la différence entre $\bar{\chi}$ et $\chi_{best}$ est inférieure à 0,14, et il est rapide dans la mesure où tous les benchmarks ont été résolus dans un temps compris entre 7 secondes et 115 secondes.

TABLEAU 4.9 – Résolution approchée des benchmarks modifiés (cas de $\chi$)

| Instance | $\chi_{best}$ | $\bar{\chi}$ | CPU | $RD_{LB}$ | $RD_{opt}$ | Instance | $\chi_{best}$ | $\bar{\chi}$ | CPU | $RD_{LB}$ | $RD_{opt}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ft06m | 9* | 9 | 16 | | | la29m | 23 | 23,6 | 33 | 9,52 | |
| ft10m | 18* | 18 | 20 | | | la30m | 23 | 23,2 | 32 | 9,52 | |
| ft20m | 24 | 24 | 19 | | 4,34 | la31m | 32* | 32 | 56 | | |
| orb01m | 18* | 18 | 18 | | | la32m | 31* | 31 | 55 | | |
| orb02m | 17* | 17 | 19 | | | la33m | 31 | 31 | 55 | | 3,33 |
| orb03m | 19 | 19 | 17 | | 5,55 | la34m | 31* | 31 | 55 | | |
| orb04m | 18* | 18 | 18 | | | la35m | 31 | 31 | 55 | 3,33 | |
| orb05m | 17* | 17 | 17 | | | la36m | 22* | 22 | 40 | | |
| orb06m | 18* | 18 | 18 | | | la37m | 23* | 23 | 40 | | |
| orb07m | 17* | 17 | 17 | | | la38m | 23 | 23 | 41 | | 4,54 |
| orb08m | 19 | 19 | 19 | | 5,55 | la39m | 22* | 22 | 40 | | |
| orb09m | 18* | 18 | 18 | | | la40m | 24* | 24 | 40 | | |
| orb10m | 17* | 17 | 17 | | | swv01m | 28 | 28,8 | 34 | 7,69 | |
| la01m | 10* | 10 | 7 | | | swv02m | 28 | 28,4 | 34 | 12,00 | |
| la02m | 11* | 11 | 7 | | | swv03m | 28 | 28,2 | 34 | 7,69 | |
| la03m | 12* | 12 | 8 | | | swv04m | 28 | 28,8 | 34 | 12,00 | |
| la04m | 11* | 11 | 8 | | | swv05m | 28 | 28,6 | 34 | 12,00 | |
| la05m | 11* | 11 | 8 | | | swv06m | 33 | 33 | 58 | 14,28 | |
| la06m | 15* | 15 | 12 | | | swv07m | 32 | 32,6 | 57 | 14,28 | |
| la07m | 16* | 16 | 12 | | | swv08m | 33 | 33,6 | 57 | 17,85 | |
| la08m | 15* | 15 | 11 | | | swv09m | 33 | 33 | 57 | 13,79 | |
| la09m | 15* | 15 | 12 | | | swv10m | 32 | 32,6 | 57 | 10,34 | |
| la10m | 16* | 16 | 12 | | | swv11m | 63 | 63,4 | 113 | 14,54 | |
| la11m | 21* | 21 | 15 | | | swv12m | 63 | 63,6 | 113 | 14,54 | |
| la12m | 20* | 20 | 15 | | | swv13m | 64 | 64 | 113 | 14,54 | |
| la13m | 20* | 20 | 15 | | | swv14m | 63 | 63,8 | 113 | 14,54 | |
| la14m | 20* | 20 | 16 | | | swv15m | 63 | 63,2 | 113 | 14,54 | |
| la15m | 20* | 20 | 14 | | | swv16m | 51* | 51 | 115 | | |
| la16m | 15* | 15 | 17 | | | swv17m | 50* | 50 | 113 | | |
| la17m | 15* | 15 | 16 | | | swv18m | 50* | 50 | 113 | | |
| la18m | 15* | 15 | 16 | | | swv19m | 50* | 50 | 113 | | |
| la19m | 15* | 15 | 16 | | | swv20m | 50* | 50 | 113 | | |
| la20m | 15* | 15 | 15 | | | abz05m | 15 | 15 | 20 | 15,38 | |
| la21m | 19* | 19 | 24 | | | abz06m | 26 | 26 | 58 | 8,33 | |
| la22m | 18* | 18 | 24 | | | abz07m | 26 | 26 | 59 | 8,33 | |
| la23m | 18* | 18 | 23 | | | abz08m | 26 | 26 | 58 | 8,33 | |
| la24m | 17* | 17 | 24 | | | abz09m | 26 | 26,4 | 59 | 8,33 | |
| la25m | 18* | 18 | 23 | | | yn01m | 28 | 28,8 | 85 | 16,66 | |
| la26m | 22* | 22 | 33 | | | yn02m | 30 | 30 | 86 | 16,00 | |
| la27m | 23 | 23 | 33 | | 4,54 | yn03m | 31 | 31 | 85 | 11,11 | |
| la28m | 22* | 22 | 33 | | | yn04m | 32 | 32,4 | 85 | 14,28 | |

* $\chi$ est optimal

#### 4.6.3.2 Instances générées

Considérons maintenant les instances de graphes mixtes générées de manière aléatoire, dont les résultats des expérimentations sont donnés dans le tableau 4.10. Les colonnes 2 et 5 (resp 3 et 6) de ce tableau représentent le pourcentage où $\chi$ est optimal (resp. le temps maximal de résolution).

Les résultats montrent bien que l'algorithme TS a résolu toutes les instances générées de manière aléatoire, et cela dans un temps relativement raisonnable et ne dépassant pas 48 secondes.

TABLEAU 4.10 – Résolution approchée des instances générées (cas de $\chi$)

| Instances | a | $\overline{\text{CPU}}$ | Instances | a | $\overline{\text{CPU}}$ |
|---|---|---|---|---|---|
| ASAA 01-10 | 100 | 44 | ASAA 31-40 | 100 | 47 |
| ASAA 11-20 | 100 | 44 | ASAA 41-50 | 100 | 47 |
| ASAA 21-30 | 100 | 44 | ASAA 51-60 | 100 | 48 |

a : Nombre de fois en pourcentage où $\chi_{best}$ est optimal

## 4.7 Expérimentations pour la coloration de graphes mixtes avec le critère $\sigma$

De manière analogue à ce que nous avons étudié précédemment pour le problème de coloration de graphes mixtes avec le critère $\chi$, nous validons en premier dans cette section les deux bornes inférieure et supérieure proposées pour le critère $\sigma$. Ensuite, nous présentons une étude expérimentale afin d'analyser la performance des méthodes exactes et de la méthode approchée proposées pour la résolution du problème de coloration de graphes mixtes avec le critère $\sigma$.

### 4.7.1 Validation des bornes proposées pour $\sigma$

Dans cette partie, nous nous intéressons à l'évaluation de la borne inférieure $LB_\sigma$ proposée. Pour cela, une étude comparative avec la borne inférieure $LB_{\sigma 1}$ d'Al-Anzi et al. (2006) est réalisée. Des expérimentations ont été effectuées sur 72 benchmarks modifiés et les résultats sont résumés dans le tableau 4.11.
Dans ce tableau, nous pouvons lire respectivement pour chaque instance, les valeurs de $LB_{\sigma 1}$, $LB_\sigma$ ainsi que $UB_{\sigma 0}$, $UB_\sigma$. Rappelons que les deux dernières valeurs représentent respectivement la borne supérieure d'Al-Anzi et al. (2006) et la borne supérieure que nous avons proposée dans la section 3.3 du chapitre 3.

D'après les résultats, nous pouvons remarquer que $LB_\sigma$ est plus efficace que $LB_{\sigma 1}$. En effet, parmi les 72 benchmarks testés, les valeurs de $LB_\sigma$ sont dans tous les cas meilleures que celles de $LB_{\sigma 1}$.

Quant à la borne supérieure $UB_\sigma$ proposée dans la section 3.3 du chapitre 3, nous pouvons aussi remarquer que toutes ces valeurs sont largement meilleures que celles retournées par la borne supérieure $UB_{\sigma 0}$ d'Al-Anzi et al. (2006).

TABLEAU 4.11 – Bornes inférieures et supérieures pour σ (benchmarks modifiés)

| Instance | $LB_{\sigma 1}$ | $LB_{\sigma}$ | $UB_{\sigma 0}$ | $UB_{\sigma}$ | Instance | $LB_{\sigma 1}$ | $LB_{\sigma}$ | $UB_{\sigma 0}$ | $UB_{\sigma}$ |
|---|---|---|---|---|---|---|---|---|---|
| ft06m | 33 | 44 | 126 | 53 | la24m | 150 | 206 | 1200 | 278 |
| ft10m | 115 | 133 | 550 | 204 | la25m | 150 | 208 | 1200 | 295 |
| ft20m | 250 | 271 | 1050 | 484 | la26m | 250 | 322 | 2100 | 467 |
| orb01m | 115 | 137 | 550 | 185 | la27m | 250 | 322 | 2100 | 468 |
| orb02m | 95 | 124 | 550 | 175 | la28m | 250 | 321 | 2100 | 444 |
| orb03m | 125 | 137 | 550 | 224 | la29m | 230 | 324 | 2100 | 475 |
| orb04m | 125 | 143 | 550 | 181 | la30m | 230 | 325 | 2100 | 478 |
| orb05m | 115 | 130 | 550 | 182 | la31m | 525 | 635 | 4650 | 966 |
| orb06m | 115 | 137 | 550 | 185 | la32m | 495 | 623 | 4650 | 939 |
| orb07m | 95 | 124 | 550 | 175 | la33m | 465 | 618 | 4650 | 933 |
| orb08m | 125 | 137 | 550 | 224 | la34m | 495 | 614 | 4650 | 913 |
| orb09m | 125 | 143 | 550 | 181 | la35m | 465 | 638 | 4650 | 969 |
| orb10m | 115 | 130 | 550 | 182 | la36m | 225 | 273 | 1800 | 355 |
| la01m | 55 | 78 | 275 | 103 | la37m | 225 | 269 | 1800 | 375 |
| la02m | 65 | 82 | 275 | 107 | la38m | 225 | 258 | 1800 | 365 |
| la03m | 75 | 88 | 275 | 107 | la39m | 225 | 259 | 1800 | 344 |
| la04m | 65 | 82 | 275 | 104 | la40m | 225 | 292 | 1800 | 368 |
| la05m | 55 | 81 | 275 | 104 | swv01m | 330 | 364 | 2100 | 727 |
| la06m | 120 | 154 | 600 | 209 | swv02m | 310 | 358 | 2100 | 705 |
| la07m | 135 | 160 | 600 | 234 | swv03m | 330 | 358 | 2100 | 710 |
| la08m | 120 | 157 | 600 | 225 | swv04m | 310 | 362 | 2100 | 726 |
| la09m | 120 | 155 | 600 | 210 | swv05m | 310 | 363 | 2100 | 699 |
| la10m | 135 | 158 | 600 | 218 | swv06m | 370 | 442 | 3150 | 803 |
| la11m | 230 | 260 | 1050 | 402 | swv07m | 370 | 437 | 3150 | 786 |
| la12m | 210 | 256 | 1050 | 379 | swv08m | 370 | 449 | 3150 | 793 |
| la13m | 210 | 259 | 1050 | 385 | swv09m | 390 | 450 | 3150 | 833 |
| la14m | 210 | 253 | 1050 | 366 | swv10m | 390 | 449 | 3150 | 770 |
| la15m | 210 | 263 | 1050 | 395 | abs05m | 75 | 117 | 550 | 159 |
| la16m | 100 | 124 | 550 | 156 | abs06m | 290 | 400 | 3150 | 558 |
| la17m | 105 | 124 | 550 | 149 | abs07m | 290 | 400 | 3150 | 558 |
| la18m | 100 | 122 | 550 | 154 | abs08m | 290 | 406 | 3150 | 554 |
| la19m | 100 | 118 | 550 | 147 | abs09m | 270 | 393 | 3150 | 574 |
| la20m | 100 | 119 | 550 | 162 | yn1m | 270 | 455 | 4200 | 600 |
| la21m | 150 | 209 | 1200 | 287 | yn2m | 270 | 462 | 4200 | 655 |
| la22m | 150 | 210 | 1200 | 297 | yn3m | 290 | 458 | 4200 | 649 |
| la23m | 150 | 203 | 1200 | 260 | yn4m | 310 | 486 | 4200 | 687 |

## 4.7.2 Résultats des méthodes exactes

Dans ce qui suit, nous abordons les résultats de l'étude comparative des trois méthodes exactes proposées. Dans une première étape, nous analysons les résultats issus des benchmarks modifiés, ensuite ceux obtenus des instances générées.

### 4.7.2.1 Benchmarks modifiés

Le tableau 4.12 synthétise les différents résultats obtenus de la résolution des benchmarks modifiés par le modèle mathématique ($MILP_{\sigma}$) ainsi que les variantes de la méthode branch and bound ($BB1_{\sigma}$ et $BB2_{\sigma}$). Notons que $BB1_{\sigma}$ utilise la borne inférieure $LB_{\sigma 1}$ proposée par Al-Anzi et al. (2006) alors que $BB2_{\sigma}$ utilise la borne inférieure $LB_{\sigma}$ que nous avons proposée.

TABLEAU 4.12 – Résolution exacte des benchmarks modifiés (cas de σ)

| Instance | $BB1_\sigma$ | | | $BB2_\sigma$ | | | $MILP_\sigma$ | |
|---|---|---|---|---|---|---|---|---|
| | σ | Noeuds | CPU | σ | Noeuds | CPU | σ | CPU |
| ft06m | 48 | 3 282 | 1 | 48 | 173 | 1 | 48 | 1 |
| ft10m | 157 | 543 195 266 | 5400 | 145 | 111 522 183 | 645 | 145 | 753 |
| ft20m | 282 | 429 840 800 | 5400 | 282 | 696 226 415 | 5400 | 280 | 5400 |
| orb01m | 153 | 550 388 457 | 5400 | 157 | 934 653 832 | 5400 | 143 | 553 |
| orb02m | 145 | 538 601 036 | 5400 | 135 | 7 263 675 | 45 | 135 | 141 |
| orb03m | 161 | 544 193 787 | 5400 | 160 | 929 827 810 | 5400 | 146 | 263 |
| orb04m | 167 | 540 299 612 | 5400 | 151 | 725 939 953 | 4251 | 153 | 5400 |
| orb05m | 139 | 539 201 183 | 5400 | 139 | 389 468 | 2 | 139 | 120 |
| orb06m | 153 | 550 388 457 | 5400 | 157 | 937 305 699 | 5400 | 143 | 553 |
| orb07m | 145 | 538 302 201 | 5400 | 135 | 7 263 675 | 45 | 135 | 141 |
| orb08m | 161 | 544 697 390 | 5400 | 160 | 933 481 399 | 5400 | 146 | 264 |
| orb09m | 167 | 540 299 612 | 5400 | 151 | 725 939 953 | 4236 | 153 | 5400 |
| orb10m | 139 | 538 103 161 | 5400 | 139 | 389 468 | 2 | 139 | 120 |
| la01m | 80 | 989 633 469 | 5400 | 80 | 17 159 279 | 79 | 80 | 7 |
| la02m | 85 | 962 287 195 | 5400 | 84 | 2 422 271 | 12 | 84 | 7 |
| la03m | 89 | 975 277 572 | 5400 | 88 | 150 867 719 | 689 | 88 | 1 |
| la04m | 83 | 950 647 908 | 5400 | 83 | 401 898 | 1 | 83 | 7 |
| la05m | 82 | 956 822 811 | 5400 | 81 | 21 001 680 | 97 | 81 | 5 |
| la06m | 165 | 748 307 123 | 5400 | 169 | 867 678 638 | 5400 | 155 | 4606 |
| la07m | 164 | 771 714 180 | 5400 | 167 | 891 009 989 | 5400 | 160 | 499 |
| la08m | 164 | 779 278 446 | 5400 | 162 | 922 146 161 | 5400 | 159 | 3645 |
| la09m | 159 | 734 591 893 | 5400 | 160 | 875 837 808 | 5400 | 155 | 1597 |
| la10m | 164 | 756 159 822 | 5400 | 168 | 864 839 844 | 5400 | 158 | 1395 |
| la11m | 286 | 563 374 029 | 5400 | 278 | 571 894 094 | 5400 | 267 | 5400 |
| la12m | 270 | 537 005 545 | 5400 | 275 | 527 804 517 | 5400 | 260 | 5400 |
| la13m | 271 | 450 916 497 | 5400 | 270 | 500 536 540 | 5400 | 260 | 5400 |
| la14m | 276 | 443 835 616 | 5400 | 274 | 478 645 448 | 5400 | 261 | 5400 |
| la15m | 276 | 629 184 391 | 5400 | 276 | 658 000 000 | 5400 | 264 | 5400 |
| la16m | 137 | 756 019 974 | 5400 | 132 | 4 699 244 | 29 | 132 | 533 |
| la17m | 134 | 767 431 532 | 5400 | 126 | 79 175 | 0 | 126 | 28 |
| la18m | 141 | 759 437 454 | 5400 | 129 | 2 107 953 | 12 | 129 | 114 |
| la19m | 126 | 4 895 731 | 35 | 126 | 55 763 | 0 | 126 | 32 |
| la20m | 132 | 760 859 100 | 5400 | 128 | 316 285 | 2 | 128 | 13 |
| la21m | 256 | 606 887 613 | 5400 | 242 | 745 309 898 | 5400 | 224 | 5400 |
| la22m | 242 | 616 543 301 | 5400 | 234 | 750 000 000 | 5400 | 225 | 5400 |
| la23m | 241 | 624 306 326 | 5400 | 230 | 751 165 372 | 5400 | 222 | 5400 |
| la24m | 232 | 614 886 132 | 5400 | 231 | 736 318 224 | 5400 | 219 | 5400 |
| la25m | 232 | 627 302 997 | 5400 | 226 | 763 575 791 | 5400 | 226 | 5400 |
| la26m | 367 | 367 048 393 | 5400 | 351 | 454 663 212 | 5400 | 361 | 5400 |
| la27m | 362 | 345 232 816 | 5400 | 366 | 460 488 346 | 5400 | 361 | 5400 |
| la28m | 383 | 377 720 207 | 5400 | 379 | 408 000 000 | 5400 | 354 | 5400 |
| la29m | 379 | 396 266 174 | 5400 | 375 | 467 826 731 | 5400 | 368 | 5400 |
| la30m | 399 | 412 084 257 | 5400 | 367 | 486 369 521 | 5400 | 379 | 5400 |
| la31m | 726 | 51 846 381 | 5400 | 702 | 52 302 632 | 5400 | 740 | 5400 |
| la32m | 749 | 34 307 497 | 5400 | 727 | 30 113 330 | 5400 | 694 | 5400 |
| la33m | 741 | 33 230 769 | 5400 | 709 | 42 535 263 | 5400 | 721 | 5400 |
| la34m | 729 | 21 263 648 | 5400 | 726 | 33 503 650 | 5400 | 726 | 5400 |
| la35m | 726 | 53 761 062 | 5400 | 712 | 43 781 095 | 5400 | 698 | 5400 |
| la36m | 331 | 599 112 426 | 5400 | 308 | 633 882 614 | 5400 | 305 | 5400 |
| la37m | 338 | 576 786 375 | 5400 | 317 | 622 538 860 | 5400 | 302 | 5400 |
| la38m | 317 | 571 788 227 | 5400 | 304 | 640 762 680 | 5400 | 292 | 5400 |
| la39m | 327 | 597 557 365 | 5400 | 316 | 630 532 939 | 5400 | 298 | 5400 |
| la40m | 362 | 581 676 846 | 5400 | 327 | 621 884 836 | 5400 | 313 | 5400 |

Dans ce tableau, les colonnes 2, 5 et 8 représentent les valeurs du critère ($\sigma$) obtenues respectivement par $BB1_\sigma$, $BB2_\sigma$ et $MILP_\sigma$, alors que les colonnes 4, 7 et 9 représentent le temps de résolution (CPU) correspondant respectivement aux $BB1_\sigma$, $BB2_\sigma$ et $MILP_\sigma$. Enfin, le nombre de noeuds (Noeuds) explorés dans l'arbre de solutions par $BB1_\sigma$ et $BB2_\sigma$ est représenté respectivement par les colonnes 3 et 6.

Concentrons nous d'abord sur l'analyse de performance du modèle ($MILP_\sigma$). Nous constatons que le modèle résout des instances de graphes d'ordre inférieur ou égal à 300 sommets. En effet, il a trouvé 25 solutions optimales parmi les 53 benchmarks testés et ce dans un délai moyen de résolution ne dépassant pas 616 secondes. Par contre l'optimalité n'est pas atteinte pour les 28 benchmarks dans le délai imposé dans cette étude.
D'autre part, pour les benchmarks pour lesquels l'optimum n'est pas connu, nous calculons le pourcentage de déviation par rapport à la borne inférieure $RD_{LB}$. Ainsi de la figure 4.6, nous constatons que $RD_{LB}$ est dans tous les cas en faveur de $MILP_\sigma$, sauf pour 4 instances des 26 benchmarks où sa valeur est de moindre qualité comparativement à celle de $BB1_\sigma$ ou de $BB2_\sigma$.

Nous remarquons aussi que les solutions obtenues par la résolution du modèle ($MILP_\sigma$) sont dans la plus part des cas de qualité meilleure ou égale à celle fournies par les méthodes $BB1_\sigma$ ou $BB2_\sigma$, excepté les deux 2 benchmarks orb04m et orb09m qui sont résolus de manière optimale uniquement par $BB2_\sigma$. Cependant, le temps de résolution (CPU) des benchmarks résolus à la fois par $MILP_\sigma$ et $BB2_\sigma$ est généralement en faveur de $BB2_\sigma$.

Concentrons nous maintenant sur les résultats des deux variantes de la méthode branch and bound $BB1_\sigma$ et $BB2_\sigma$. Nous remarquons que $BB1_\sigma$ et $BB2_\sigma$ ont aussi résolu des benchmarks d'ordre inférieur ou égal à 300 sommets, alors que l'optimum n'est pas atteint dans le délai imposé pour les benchmarks d'ordre supérieur à 100 sommets.

De manière générale, nous constatons aussi que $BB2_\sigma$ est généralement plus performante que $BB1_\sigma$ de point de vue de :

- nombre de solutions optimales obtenues ($N_{opt}$) : parmi les 53 benchmarks testés, $BB2_\sigma$ retourne 18 solutions optimales, alors que $BB1_\sigma$ n'a trouvé que 2 solutions optimales. A ce niveau, nous n'avons pas considéré les solutions optimales déduites de la comparaison avec $MILP_\sigma$ (i.e. les solutions obtenues, dans la limite du temps imposé, sans que les noeuds de l'arbre de solutions soient totalement stérilisés). Il s'agit précisément des benchmarks orb05m, orb10m , la01m et la04m résolus par $BB1_\sigma$.
- temps de résolution optimale des benchmarks (CPU) : dans tous les cas où les benchmarks sont résolus de manière optimale par les deux méthodes $BB1_\sigma$ et $BB2_\sigma$, ce temps est en faveur de $BB2_\sigma$.
- nombre de noeuds explorés au niveau de l'arbre de solutions (Noeuds) : pour les benchmarks résolus de manière optimale par $BB1_\sigma$ et $BB2_\sigma$, ce temps est dans la plupart des cas en faveur de $BB2_\sigma$.

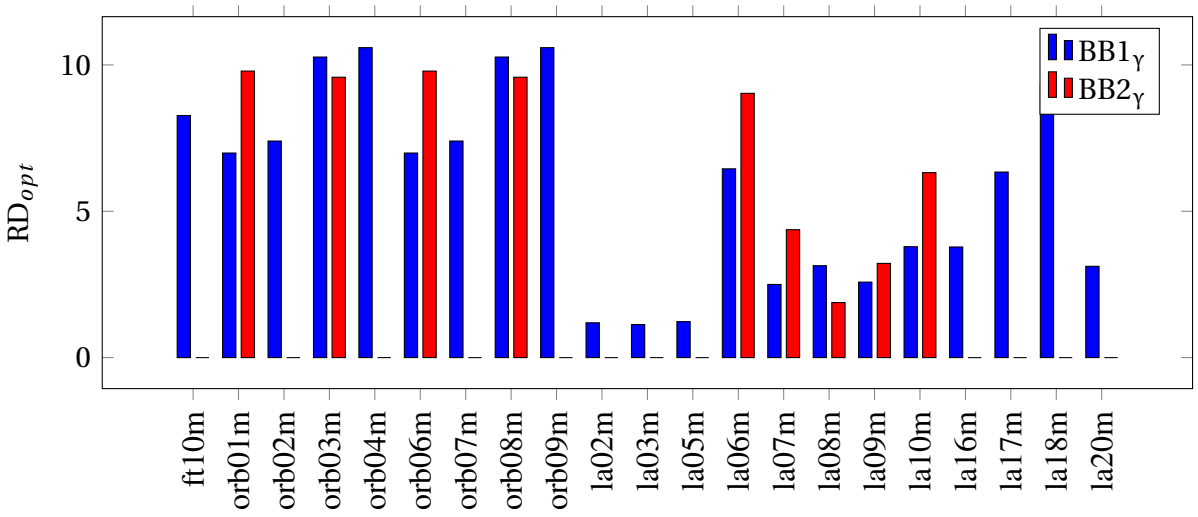


FIGURE 4.5 – Déviation par rapport à l'optimum pour les benchmarks modifiés (cas de σ)

– pourcentage de déviation par rapport à la solution optimale (RD$_{opt}$) : ce pourcentage est calculé pour les benchmarks résolus de manière non-optimale par au moins une des deux méthodes (BB1$_\sigma$ ou BB2$_\sigma$), tout en supposant que leur solution optimale est connue. Pour cet indicateur de performance, nous remarquons d'une part que RD$_{opt}$ n'excède pas 9,79% (resp. 10,59%) pour BB2$_\sigma$ (resp. BB1$_\sigma$) (cf. figure 4.5), et d'autre part que ces pourcentages sont majoritairement en faveur de BB2$_\sigma$ (i.e. 15 meilleures valeurs pour BB2$_\sigma$ vs 6 pour BB1$_\sigma$).

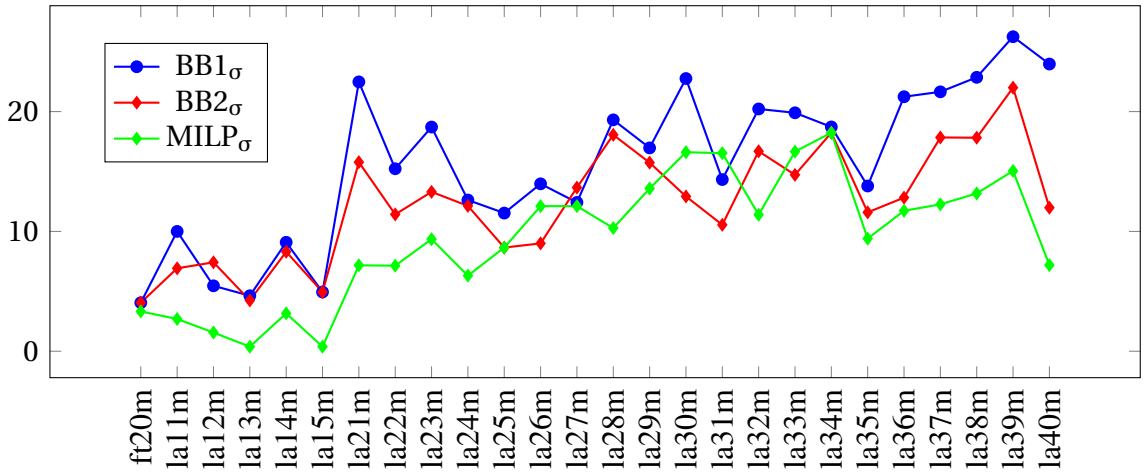


FIGURE 4.6 – Déviation par rapport à la borne inférieure pour les benchmarks modifiés (cas de σ)

– pourcentage de déviation par rapport à la borne inférieure (RD$_{\text{LB}}$) : cette valeur est calculée quand la solution optimale n'est obtenue par aucune des 3 méthodes exactes. De la figure 4.6, nous constatons que RD$_{\text{LB}}$ est en faveur de BB2$_\sigma$ pour l'ensemble de benchmarks concernés, excepté les deux benchmarks la12m et la27 où RD$_{\text{LB}}$ est en faveur de BB1$_\sigma$.

#### 4.7.2.2 Instances générées

A l'inverse des benchmarks modifiés vérifiant la propriété que "tout chemin de longueur maximale visite exactement une seule fois une clique maximale", les instances générées peuvent avoir des chemins de longueur maximale pouvant soit ne pas visiter au moins une clique maximale, ou bien visiter une même clique plus d'une fois. Pour ce type d'instances, les deux bornes inférieures $LB_\sigma$ et $LB_{\sigma 1}$ ne sont pas valides (cf. le contre-exemple présenté dans la section 2.4.3 du chapitre 2 et l'hypothèse de la section 3.2.2 du chapitre 3).
Par conséquent, nous nous limitons à étudier les résultats de $MILP_\sigma$ sur les benchmarks modifiés, où les résultats sont présentés dans le tableau 4.13. Les colonnes 2 et 5 (resp. 3 et 6) de ce tableau représentent le pourcentage où σ est optimal (resp. le temps moyen de résolution).

Nous remarquons que pour les 60 instances testées, $MILP_\sigma$ a retourné 54 solutions optimales. Le temps moyen de résolution de ces instances est compris entre 635,2 secondes et 1127,5 secondes pour les trois premiers types d'instances, et il est compris entre 108,3 secondes et 484,6 secondes pour les trois autres types d'instances.

TABLEAU 4.13 – Résolution exacte des instances générées (cas de σ)

| Instances | MILP | | Instances | MILP | |
|---|---|---|---|---|---|
| | a | $\overline{CPU}$ | | a | $\overline{CPU}$ |
| ASAA 01-10 | 100 | 1127,5 | ASAA 31-40 | 100 | 171,1 |
| ASAA 11-20 | 70 | 1599,3 | ASAA 41-50 | 100 | 484,6 |
| ASAA 21-30 | 90 | 635,2 | ASAA 51-60 | 100 | 108,3 |

a : Nombre de fois en pourcentage où σ est optimal

### 4.7.3 Résultats de l'algorithme TS

Partant du fait que les deux problèmes de coloration de graphes ont les mêmes solutions réalisables, les paramètres de l'algorithme TS fixés pour la résolution du problème de coloration de graphes mixtes avec le critère χ sont ainsi considérés pour l'optimisation du critère σ (i.e. le nombre maximal d'itérations $Max-iter = 1000$, la taille de la liste tabu $list = n \times m$, la taille du voisinage $Vs = 1000$, le nombre d'iterations utilisées dans la structure de voisinage $l = 100$).

Dans la suite de cette section, nous procédons à l'analyse des résultats de l'algorithme TS sur les deux types d'instances (benchmarks modifiés et instances générées), sachant que l'exécution de chacune des instances du problème a été lancée exactement 5 fois.

#### 4.7.3.1 Benchmarks modifiés

De la même manière que pour la résolution du problème de coloration de graphes mixtes avec le critère χ, la performance de l'algorithme TS est évaluée en utilisant les trois mesures :

TABLEAU 4.14 – Résolution approchée des benchmarks modifiés (cas de σ)

| Instance | $\sigma_{best}$ | $\bar{\sigma}$ | CPU | $RD_{LB}$ | $RD_{opt}$ | Instance | $\sigma_{best}$ | $\bar{\sigma}$ | CPU | $RD_{LB}$ | $RD_{opt}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ft06m | 48* | 48,0 | 16 | | | la24m | 219 | 220,4 | 29 | 6,31 | |
| ft10m | 146 | 148,2 | 25 | | 0,69 | la25m | 229 | 231,4 | 29 | 10,10 | |
| ft20m | 278 | 280,4 | 24 | 2,58 | | la26m | 344 | 349,4 | 42 | 6,83 | |
| orb01m | 146 | 147,6 | 22 | | 2,10 | la27m | 347 | 351,8 | 42 | 7,7 | |
| orb02m | 135* | 136,6 | 26 | | | la28m | 348 | 352,2 | 42 | 8,41 | |
| orb03m | 149 | 151,0 | 26 | | 2,05 | la29m | 349 | 356,2 | 42 | 7,72 | |
| orb04m | 153 | 153,6 | 26 | | | la30m | 351 | 358,2 | 42 | 8,00 | |
| orb05m | 140 | 140,4 | 26 | | 0,72 | la31m | 669 | 681,6 | 74 | 5,35 | |
| orb06m | 144 | 147,4 | 26 | | 0,70 | la32m | 673 | 678,8 | 75 | 8,03 | |
| orb07m | 135* | 136,6 | 26 | | | la33m | 675 | 685,6 | 75 | 9,22 | |
| orb08m | 148 | 149,8 | 25 | | 1,37 | la34m | 672 | 682,4 | 75 | 9,45 | |
| orb09m | 151* | 154,2 | 26 | | | la35m | 683 | 689,0 | 75 | 7,05 | |
| orb10m | 139* | 139,8 | 26 | | | la36m | 294 | 300,0 | 51 | 7,69 | |
| la01m | 80* | 80,6 | 8 | | | la37m | 304 | 306,4 | 51 | 13,01 | |
| la02m | 84* | 84,8 | 9 | | | la38m | 295 | 299,8 | 51 | 14,34 | |
| la03m | 88* | 88,0 | 9 | | | la39m | 292 | 296,0 | 51 | 12,74 | |
| la04m | 83* | 83,0 | 9 | | | la40m | 311 | 313,4 | 52 | 6,51 | |
| la05m | 81* | 81,2 | 9 | | | swv01m | 398 | 403,0 | 44 | 9,34 | |
| la06m | 157 | 159,2 | 12 | | 1,29 | swv02m | 403 | 410,8 | 44 | 12,57 | |
| la07m | 161 | 161,8 | 13 | | 0,63 | swv03m | 406 | 409,4 | 44 | 13,41 | |
| la08m | 158 | 159,2 | 13 | 0,64 | | swv04m | 405 | 413,2 | 44 | 11,88 | |
| la09m | 156 | 157,6 | 12 | | 0,65 | swv05m | 398 | 407,0 | 44 | 9,64 | |
| la10m | 159 | 160,0 | 12 | | 0,63 | swv06m | 517 | 532,0 | 75 | 16,97 | |
| la11m | 265 | 267,6 | 17 | 1,92 | | swv07m | 523 | 529,6 | 75 | 19,68 | |
| la12m | 260 | 262,6 | 17 | 1,56 | | swv08m | 522 | 531,8 | 75 | 16,26 | |
| la13m | 261 | 262,6 | 17 | 0,77 | | swv09m | 525 | 536,8 | 75 | 16,67 | |
| la14m | 258 | 259,2 | 17 | 1,98 | | swv10m | 522 | 529,2 | 75 | 16,26 | |
| la15m | 267 | 268,0 | 17 | 1,52 | | abz05m | 129 | 130,0 | 26 | 10,26 | |
| la16m | 132* | 133,6 | 19 | | | abz06m | 442 | 447,6 | 78 | 10,50 | |
| la17m | 126* | 127,0 | 19 | | | abz07m | 440 | 447,2 | 78 | 10,00 | |
| la18m | 129* | 130,0 | 19 | | | abz08m | 443 | 449,2 | 78 | 9,11 | |
| la19m | 128 | 128,0 | 19 | | 1,59 | abz09m | 447 | 450,6 | 78 | 13,74 | |
| la20m | 128* | 128,0 | 19 | | | yn01m | 521 | 522,6 | 117 | 14,51 | |
| la21m | 227 | 228,2 | 29 | 8,61 | | yn02m | 531 | 535,8 | 117 | 14,94 | |
| la22m | 225 | 230,6 | 29 | 7,14 | | yn03m | 534 | 536,6 | 117 | 16,59 | |
| la23m | 219 | 222,8 | 29 | 7,88 | | yn04m | 549 | 555,4 | 118 | 12,96 | |

* σ est optimal

la meilleure solution trouvée des cinq lancements de la même instance ($\sigma_{best}$), la moyenne du coût de ces solutions ($\bar{\sigma}$) et le temps maximal de résolution (CPU).
Une analyse des résultats présentés dans le tableau 4.14 montre que l'algorithme TS retourne des solutions optimales pour 14 benchmarks parmi les 72 benchmarks testés (résultat moins efficace comparativement à son adaptation pour le critère χ).

Pour les solutions non optimales, l'indicateur sur la moyenne de $RD_{opt}$ (resp. $RD_{LB}$) est évaluée pour $\sigma_{best}$ lorsque la solution optimale est connue (resp. inconnue).
A partir de ces deux mesures (la moyenne de $RD_{opt}$ = 1,12% et la moyenne de $RD_{LB}$ = 9,53%), nous déduisons que les solutions approchées sont généralement de bonne qualité.

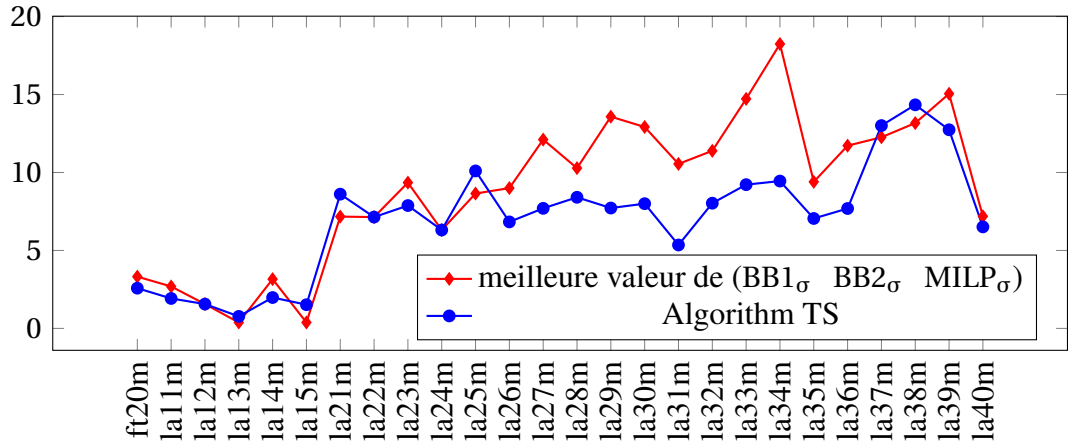


FIGURE 4.7 – Déviation par rapport à la borne inférieure pour les benchmarks modifiés (cas de σ) : meilleure valeur de ($BB1_\sigma$, $BB2_\sigma$, $MILP_\sigma$) vs Algorithme TS

.

Concentrons maintenant sur les benchmarks résolus de manière non optimale. Pour ces benchmarks, nous considérons la meilleure valeur de $RD_{LB}$ issue des 3 méthodes exactes (cf. figure 4.7). Nous comparons ensuite cette meilleure valeur avec $RD_{LB}$ de l'algorithme TS. De la figure 4.7, nous remarquons que l'algorithme TS donne des résultats plus efficaces comparativement aux meilleurs résultats des 3 méthodes exactes. En effet, l'algorithme TS donne 17 meilleures solutions vs 6 meilleures solutions obtenues par une des méthodes exactes proposées. Néanmoins, nous pouvons constater que l'algorithme TS est assez rapide puisque tous les benchmarks sont résolus dans un CPU compris entre 8 secondes et 118 secondes, et est robuste puisque la moyenne (resp. le maximum) des pourcentages de l'écart entre $\bar{\sigma}$ et $\sigma_{best}$ est égale à 1,13% (resp. 2,9%).

#### 4.7.3.2 Instances générées

Le tableau 4.15 présente les résultats de l'algorithme TS exécuté sur les instances de graphes mixtes générées de manière aléatoire. Dans ce tableau, les colonnes 2 et 6 (resp 3 et 7) représentent le pourcentage où σ est optimal (resp. le temps maximal de résolution). Nous pouvons constater que l'algorithme TS a résolu de manière optimale toutes les instances du $6^{ième}$ type, uniquement 20% des instances du $2^{ième}$ type, et entre 60% et 80% des autres types d'instances. Nous remarquons aussi que toutes les instances sont résolues en un temps relativement raisonnable ne dépassant pas 51 secondes.

Cependant, pour les solutions non optimales du même type d'instances, la moyenne de $RD_{opt}$ est calculée uniquement lorsque la solution optimale est obtenue par $MILP_\sigma$, et cette valeur est précisée dans les colonnes 4 et 8 de ce tableau.

TABLEAU 4.15 – Résolution approchée des instances générées (cas de σ)

| Instances | a | $\overline{CPU}$ | $RD_{opt}$ | Instances | a | $\overline{CPU}$ | $RD_{opt}$ |
|---|---|---|---|---|---|---|---|
| ASAA01-10 | 80 | 46 | 0,64 | ASAA31-40 | 70 | 46 | 1,42 |
| ASAA11-20 | 20 | 46 | 0,85 | ASAA41-50 | 60 | 51 | 0,63 |
| ASAA21-30 | 80 | 47 | 0,44 | ASAA51-60 | 100 | 46 | |

a : Nombre de fois en pourcentage où σ est optimal

## 4.8 Conclusion

Dans ce chapitre nous avons procédé à une étude expérimentale pour l'évaluation de la performance des bornes et des méthodes de résolution proposées pour les deux problèmes de coloration de graphes mixtes. Le premier problème de coloration s'intéresse au nombre chromatique (χ) et le second au critère de minimisation de la somme des couleurs des extrémités terminales des chemins de longueur maximale (σ). Nous rappelons que le premier problème de coloration modélise le problème d'ordonnancement job shop avec des durées unitaires minimisant le makespan ($J|p_{ij} = 1|C_{max}$), et le second modélise le problème d'ordonnancement job shop avec des durées unitaires minimisant la somme des dates de fin des jobs ($J|p_{ij} = 1|\sum C_i$).

Rappelons que ces deux problèmes ont été déjà abordés dans la littérature par la proposition de méthodes exactes pouvant résoudre de manière optimale des instances de graphes mixtes de 200 sommets ou moins. Étant donné l'inexistence de benchmarks permettant de comparer et mesurer la force et la puissance des méthodes que nous avons proposées relativement aux méthodes de la littérature, ce constat nous a motivé à adapter d'une part les benchmarks du job shop d'OR-Library aux problèmes étudiés, et d'autre part de générer des instances par le mécanisme de génération d'instances adopté de la littérature.

Ensuite, afin de confirmer l'hypothèse que la difficulté des instances pour ce type de problèmes ne dépend pas strictement de la taille des instances, nous avons présenté quelques indicateurs permettant d'identifier les instances faciles et les instances difficiles.

Le schéma de validation des bornes (inférieures et supérieures) et des méthodes de résolution proposées est scindé en deux principales parties : expérimentations pour la coloration de graphes mixtes avec le critère χ et expérimentations pour la coloration de graphes mixtes avec le critère σ.

Pour chacun des deux problèmes, nous avons utilisé les deux types d'instances pour analyser :

- les résultats de la comparaison des bornes proposées avec les bornes de la littérature.
- les résultats des 3 méthodes exactes, à savoir les deux variantes de la méthode branch and bound $BB1_\chi$ et $BB2_\chi$ (resp. $BB1_\sigma$ et $BB2_\sigma$) proposées pour la coloration de graphes mixtes avec le critère χ (resp. σ), ainsi que la méthode basée sur la modélisation mathématique $MILP_\chi$ (resp. $MILP_\sigma$) avec le critère χ (resp. σ).

– les résultats de la méthode approchée basée sur l'algorithme TS.

A la base d'une étude approfondie sur les différents résultats obtenus nous avons constaté que :

– En se basant sur une configuration efficace du solveur CPLEX, la méthode de résolution basée sur la modélisation mathématique $MILP_\chi$ (resp. $MILP_\sigma$) a donné, dans la plupart des cas et dans la limite du temps de résolution imposé, des meilleurs résultats comparativement aux résultats obtenus par les méthodes branch and bound $BB1_\chi$ et $BB2_\chi$ (resp. $BB1_\sigma$ et $BB2_\sigma$). Néanmoins, pour un temps de résolution plus étendu, $BB2_\chi$ a réussi à résoudre de manière optimale 2 des 3 benchmarks que $MILP_\chi$ a échoué à résoudre en 86400 secondes.

– La méthode $BB2_\chi$ utilisant la borne inférieure $LB_\chi$ proposée (resp. $BB2_\sigma$ utilisant la borne inférieure $LB_\sigma$ proposée) est généralement plus performante que la méthode $BB1_\chi$ utilisant la borne inférieure $LB_{\chi 1}$ de la littérature (resp. $BB1_\sigma$ utilisant la borne inférieure $LB_{\sigma 1}$ de la littérature).

– L'algorithme TS adapté pour les deux problèmes de coloration de graphes mixtes a obtenu de meilleurs résultats pour la plupart des deux types d'instances. Sa performance est évaluée en retenant la meilleure solution trouvée des cinq lancements d'une même instance, la moyenne du coût de ces solutions et le maximum de leur temps de résolution.

– La borne inférieure $LB_\chi$ proposée pour $\chi$ est meilleure que celle issue de la littérature $LB_{\chi 1}$, et elle est efficace dans la mesure où elle coïncide avec la solution optimale pour 29 instances parmi les 82 benchmarks modifiés et pour 27 des 60 instances générées.

  De même, la borne inférieure $LB_\sigma$ proposée pour $\sigma$ est meilleure que celle de la littérature $LB_{\sigma 1}$. Par contre, elle ne coïncide avec la solution optimale que pour 2 instances parmi les 72 benchmarks modifiés.

– La borne supérieure $UB_\chi$ (resp. $UB_\sigma$) proposée pour $\chi$ (resp. $\sigma$) est, dans tous les cas, meilleure que la borne supérieure de la littérature $UB_{\chi 1}$ (resp. $UB_{\sigma 1}$) proposée pour $\chi$ (resp. $\sigma$).

Enfin nous rappelons que parmi les points forts des méthodes branch and bound proposées ($BB1_\chi$,$BB2_\chi$, $BB1_\sigma$ et $BB2_\sigma$), nous citons leur aptitude de gérer efficacement la mémoire principale en optimisant la taille de l'arbre et en ne gardant que les noeuds pendants. De même, le nombre élevé de noeuds générés par chaque méthode dans la limite de 5400 secondes témoigne du bon choix de la stratégie de recherche utilisée dans le parcours de l'arbre de solutions.

# Conclusion générale

Le travail présenté dans cette thèse s'intéresse aux problèmes d'ordonnancement par la coloration de graphes, et précisément à deux problèmes d'ordonnancement de type job shop avec des opérations de durée unitaire. Le premier problème ($J|p_{ij} = 1|C_{max}$) traite l'optimisation du makespan, alors que le second ($J|p_{ij} = 1|\sum C_i$) traite celle de la somme des dates de fin d'exécution des jobs.

Chacun des deux problèmes d'ordonnancement est modélisé par la coloration d'une classe spéciale de graphes mixtes caractérisée par : le graphe partiel engendré par l'ensemble des arêtes constitue une union disjointe de cliques maximales, et le graphe partiel engendré par les arcs constitue une union disjointe de chemins de longueur maximale.

Ainsi, deux problèmes de coloration de graphes mixtes sont distingués. Le premier s'intéresse au nombre chromatique (noté $\chi$), et le second optimise la somme des couleurs des extrémités terminales des chemins de longueur maximale (notée $\sigma$).

L'étude de ces problèmes est motivée par l'existence de nombreuses applications pratiques ayant fait objet de plusieurs travaux dans la littérature. De plus, ces problèmes sont déjà abordés dans la littérature par la proposition des bornes inférieures et supérieures pouvant être améliorées, et par la proposition d'algorithmes branch and bound limités par la taille de l'arbre de solutions (ne dépassant pas les 20 000 000 noeuds).

Dans le cadre des travaux menés dans cette thèse, nos contributions portent sur plusieurs aspects résumés par les points suivants :

- *Proposition de bornes inférieures pour les deux critères à optimiser* : Une borne inférieure $LB_\chi$ (resp. $LB_\sigma$) est proposée pour le nombre chromatique $\chi$ (resp. $\sigma$). Ces bornes sont basées sur la coloration optimale de $m$ graphes partiels, où chaque graphe partiel est engendré par l'ensemble des arcs et les arêtes d'une clique maximale.
- *Proposition de bornes supérieures pour les critères à optimiser* : une borne supérieure $UB_\chi$ (resp. $UB_\sigma$) pour le nombre chromatique $\chi$ (resp. $\sigma$) est proposée en adaptons l'algorithme DSATUR à la coloration de graphes mixtes.
- *Proposition de programmes linéaires à variables mixtes (Mixed Integer Linear Programs MILP)* : un modèle $MILP_\chi$ (resp. $MILP_\sigma$) est proposé pour la modélisation du problème de coloration de graphes mixtes avec le critère $\chi$ (resp. $\sigma$ ). Ces modèles sont basés sur

l'indexation de la couleur au niveau des variables décisionnelles, et sont caractérisés par le nombre important de variables résultant du produit du nombre de sommets et le nombre de couleurs à utiliser. La mise en oeuvre efficace de ces modèles avec un nombre fini de variables décisionnelles nécessite l'utilisation d'un nombre limité de couleurs. Pour ce faire, ce nombre est contraint par la valeur de la borne supérieure $UB_\chi$ proposée.

- *Proposition d'une méthode de résolution basée sur branch and bound* : l'efficacité d'une méthode par séparation et évaluation dépend étroitement du choix de la stratégie à adopter dans chacune des trois principales composantes constituant cette méthode, à savoir la stratégie de recherche, la stratégie de séparation et la stratégie d'élagage. Ces composantes influent de manière directe sur la performance de ce type de méthodes. Ainsi, un bon choix de ces stratégies permet de gérer efficacement le problème du surdimensionnement de l'arbre de solutions, et de guider rapidement le processus de recherche vers une solution optimale.

  Dans la méthode proposée pour la résolution des deux problèmes de coloration de graphes mixtes, les trois composantes constituants la méthode branch and bound sont développées comme suit :

  – Le codage d'une solution adopté se diffère de celui de la littérature par l'ajout de deux autres vecteurs de données, ceci a permis d'évaluer une solution partielle d'un sous-problème associé à un noeud sans avoir recours à ses ascendants.

  – Dans la stratégie de recherche, nous avons adopté le parcours DFS, dans lequel nous sélectionnons le noeud possédant la meilleure borne inférieure. Ceci a permis d'explorer plus intelligemment l'arbre de solutions et d'éviter l'utilisation excessive de la mémoire.

  – Dans les règles d'élagage, nous avons adopté deux bornes inférieures pour chaque problème de coloration, faisant ainsi deux variantes de l'algorithme branch and bound pour chacun des deux problèmes.

  Tirant profit du bon choix de ces trois composantes, et partant de l'idée que les solutions complètes se trouvent au niveau des noeuds pendants de l'arbre de solutions, le mécanisme que nous avons proposé a effectivement permis de ne manipuler que les noeuds pendants tout en éliminant les autres noeuds de l'arbre de solutions. En terme de résultats, ce mécanisme a permis de libérer l'espace excessif de stockage et aussi d'optimiser la taille de mémoire réservée à l'arbre de solutions. De plus, la méthode branch and bound a réussi d'explorer plus de 989 633 460 noeuds de l'arbre de solutions.

- *Proposition d'une méthode approchée basée sur l'algorithme tabu search* : pour le cas des instances de problèmes de grande taille, nous avons proposé une méthode approchée basée sur l'algorithme tabu search en utilisant une structure dynamique de voisinage.

Pour la validation et l'analyse de la performance des bornes et des méthodes de résolution proposées, des expérimentations ont été menées en utilisant des benchmarks modifiés de la littérature et des instances générées. Les résultats issus de ces expérimentations démontrent empiriquement l'efficacité des méthodes de résolution proposées, et attestent la bonne qualité des bornes inférieures et supérieures proposées comparativement à celles proposées dans la littérature.

En perspectives, les travaux réalisées dans le cadre de cette thèse ouvrent la voie à d'autres études qui peuvent être menées dans les directions suivantes :

- Les variantes d'algorithmes branch and bound peuvent être améliorées en adoptant dans une stratégie de recherche distribuée par meilleure valeur (DBeFS). C'est une stratégie hybride entre la recherche utilisant le parcours en profondeur (DFS) et le parcours par meilleure valeur (BeFS), permettant ainsi de surmonter leurs inconvénients et d'exploiter leurs avantages.
- Les méthodes exactes et approchées sont proposées pour la coloration de graphes mixtes optimisant deux objectifs distincts. Il serait toutefois intéressant de les développer pour la coloration de graphes mixtes bi-objectives.
- Ces problèmes peuvent aussi être abordés par des algorithmes approchés qui fournissent, en temps polynomial, des solutions approchées dont l'écart à l'optimum peut être borné supérieurement.

# Bibliographie


Achterberg, T., Koch, T., and Martin, A. (2005). Branching rules revisited. *Operations Research Letters*, 33(1) :42–45. 54

Adams, J., Balas, E., and Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3) :391–401. 68

Al-Anzi, F. S., Sotskov, Y. N., Allahverdi, A., and Andreev, G. V. (2006). Using mixed graph coloring to minimize total completion time in job shop scheduling. *Applied Mathematics and Computation*, 182 :1137–1148. 26, 32, 45, 58, 61, 68, 83, 84

Allen, J. (1991). Time and time again : The many ways to represent time. *International Journal of Intelligent Systems*, 6(4) :341–355.

Appel, K. and Haken, W. (1976). Every planar map is four colorable. *Part I : Discharging. Illinois Journal of Mathematics*, 21 :429–490. 8, 15

Applegate, D. and Cook, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on computing*, 3(2) :149–156. 68

Barthélémy, J. P., Cohen, G., and Lobstein, A. (1992). Complexité algorithmique et problèmes de communication. *Collection technique et scientifique des télécommunications. Masson.* 21

Berge, C. (1960). Les problèmes de coloration en théorie des graphes. *Publications de l'Institut de Statistiques, Université Paris 9*, pages 123–160. 14

Berge, C. (1997). Motivation and history of some of my conjectures. *Discrete Mathematics*, 165 :61–70.

Berge, C. and Chvatal, V. (1984). Topics on perfect graphs. *Annals of Discrete Mathematics 21. North Holland, Amsterdam.* 14

Bondy, J. and Murty, U. (2008). Graph theory. *Springer.* 12

Bondy, J. and Murty, U. S. R. (1976). Graph theory with applications. *Elsevier science publishing*.

Brélaz, D. and Zabala, P. (1979). New methods to color the vertices of a graph. *Communications of the ACM*, 22(4) :251–256. 45

Brooks, R. L. (1941). On colouring the nodes of a network. *Proc. Cambridge Philosophical Society, Math. Phys. Sci*, 37 :194–197. 14

Brucker, P. (1999). Preemptive job-shop scheduling problems with a fixed number of jobs. *Mathematical Methods of Operations Research*, 49 :41–76. 29

Carlier, J. (1982). The one-machine sequencing problem. *European Journal of Operational Research*, 11(1) :42–47. 54

Carlier, J. and Chrétienne, P. (1988). Problèmes d'ordonnancement : mod élisation/complexité/algorithmes. *Masson*. 15

Cook, S. A. (1971). The complexity of theorem proving procedures. in association of computing machinery. *New York : editor, Proceedings of the third annual ACM symposium on the theory of computing*, pages 151–158. 21

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2009). Introduction to algorithms. *The MIT Press*, Third Edition. 52

Diestel, R. (2000). Graph theory. *Springer, second edition. New York*, 57. 12

Esquirol, P. and Lopez, P. (1999). L'ordonnancement. *Economica*. 15

Fisher, H. and Thompson, G. L. (1963). Probabilistic learning combinations of local jobshop scheduling rules. *Industrial scheduling*, 3(2) :225–251. 68

Furmanczyk, H., Kosowski, A., and Zylinski, P. (2008). A note on mixed tree coloring. *Information Processing Letters*, 106(4) :133–135. 30

Garey, M. and Johnson, D. (1979). Computer and intractability : A guide to the theory of np-completeness. *W.H. Freeman and Co, San Francisco, CA*. 21, 22, 23, 30

Gerstl, E. and Mosheiov, G. (2014). A two-stage flexible flow shop problem with unit-execution-time jobs and batching. *International Journal of Production Economics*, 158 :171–178. 31

Glover, F. (1989). Tabu search. *Part I, ORSA Journal on Computing*, 1(3) :190–206. 62, 63

Glover, F. (1990). Tabu search. *Part II, ORSA Journal on Computing*, 2(1) :4–32. 62

Golumb, M. (2004). Algorithmic graph theory and perfect graphs (graphs second edition). *Annals of discrere mathematics, Elsevier*, 57. 12

Gonzales, T. and Sahni, S. (1978). Flowshop and jobshop schedules : complexity and approximation. *Operations Research*, 263 :36–52. 29

Gonzalez, T. (1982). Unit execution time shop problems, mathematics of operations research. *Mathematics of Operations Research*, 7(1) :57–66. 29

Guschinskaya, O. (2007). Outils d'aide à la décision pour la conception en avant-projet des systèmes d'usinage à boîtiers multibroches. *Génie des procédés. Ecole Nationale Supérieure des Mines de Saint-Etienne*, <NNT : 2007EMSE0021>(N° d'ordre : 454 GI). 28

Hansen, P., Kuplinsky, J., and de Werra, D. (1997). Mixed graph colorings. *Mathematical Methods of Operations Research*, 45(1) :145–160. 30

Hefetz, N. and Adiri, I. (1982). An efficient optimal algorithm for the two-machines unit-time jobshop schedule-length problem. *Mathematics of Operations Research*, 7(3) :354–360. 29

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2010). Automated configuration of mixed integer programming solvers. *In International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 186–202. 70, 71

Ibaraki, T. (1976). Theoretical comparisons of search strategies in branch-and-bound algorithms. *International Journal of Computer and Information Sciences*, 5(4) :315–344. 50

Jackson, J. (1956). An extension of johnson's results on job lot scheduling. *Naval Research Logistics. Quart*, 3 :201–203. 29

Jain, A. and Meeran, S. (1999). Deterministic job-shop scheduling : Past, present and future. *European Journal of Operational Research*, 113 :390–434. 68, 70

Kao, G., Sewell, E., and Jacobson, S. (2009). A branch, bound and remember algorithm for the 1|rilsum ti scheduling problem. *Journal of Scheduling*, 12 :163–175. 53

Karp, R. (1972). Reducibility among combinatorial problems. *Plenum Press, NewYork, NY*. 14, 21

Kouider, A. (2014). Coloration de graphes mixtes et ses applications aux problèmes d'ordonnancement job-shop. *Memoire de Magister en mathématiques, spétialité Recherche Opérationnelle*. 31

Kouider, A., Ait Haddadène, H., Ourari, S., and Oulamara, A. (2014). Coloration de graphes mixtes et son application à l'ordonnancement job shop. *10th International Conference on Modeling, Optimization and SIMulation (MOSIM 2014). Nancy, France. November 05 to 07*. 31

Kouider, A., Ait Haddadène, H., Ourari, S., and Oulamara, A. (2015). Mixed integer linear programs and tabu search approach to solve mixed graph coloring for unit-time job shop scheduling. *The eleventh annual IEEE International Conference on Automation Science and Engineering (IEEE CASE 2015). Gothenburg, Sweden. August 24 to 28,*, pages 1177–1181. 31

Kouider, A., Ait Haddadène, H., Ourari, S., and Oulamara, A. (2017). Mixed graph colouring for unit-time scheduling. *Journal International Journal of Production Research*, 55(6) :1720–1729. 26, 31

Land, A. and Doig, A. (1960). An automatic method for solving discrete programming problems. *Econometrica*, 28(3) :497–520. 49

Lawrence, S. (1984). Resource constrained project scheduling an experimental investigation of heuristic scheduling techniques (supplement). *Graduate School of Industrial Administration*. 68

Lenstra, J. K. and Rinnooy Kan, A. H. G. (1979). Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 4 :121–140. 29

Lovasz, L. (1972). Normal hypergraphs and the perfect graph conjecture. *Discrete Mathematics*, 2 :253–267. 14

Méndez-Díaz, I. and Zabala, P. (2006). A branch and cut algorithm for graph coloring. *Discrete Applied Mathematics*, 154(5) :826–847. 45

Morrison, D., Jacobson, S., Sauppe, J., and Sewell, E. (2016). Branch-and-bound algorithms : A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19 :79–102. 52, 54, 55

Moukrim, A., Rebaine, D., and Serairi, M. (2014). A branch and bound algorithm for the two-machine flowshop problem with unit-time operations and time delays. *RAIRO-Operations Research*, 48 :235–254. 31

Oulamara, A. (2009). Contribution à l'étude des problèmes d'ordonnancement flowshop avec contraintes supplémentaires : Complexité et méthodes de résolution. *HDR de Institut National Polytechnique de Lorraine - INPL*. 21

Pinedo, M. (2008). Scheduling : Theory, algorithms, and systems. *Prentice Hall Third Edition*. 15, 16

Rebaine, D. (2010). Scheduling flexible flowshops with unit-time operations and minimum time delays. *Electronic Notes in Discrete Mathematics*, 36(1) :1193–1200. 31

Ries, B. (2007). Coloring some classes of mixed graphs. *Discrete Applied Mathematics*, 155(1) :1–6. 30

Ries, B. and de Werra, D. (2008). On two coloring problems in mixed graphs. *European Journal of Combinatorics*, 29 :712–725. 30

Scholl, A. and Klein, R. (1999). Balancing assembly lines effectively a computational comparison. *European Journal of Operational Research*, 114 :50–58. 50

Sotskov, Y. (1985). Optimal scheduling two jobs with regular criterion. in : Design process automating. *Institute of Engineering Cybernetics, Minsk, Belarus*, pages 85–95. 29

Sotskov, Y. (1991). The complexity of shop-scheduling problems with two or three jobs. *European Journal of Operational Research*, 53. 29

Sotskov, Y., Dolgui, A., and Werner, F. (2001). Mixed graph colouring for unit-time job-shop scheduling. *International Journal of Mathematical Algorithms*, 2(4) :289–323. 26, 28, 29, 31, 32, 42, 58, 61, 73, 74, 76

Sotskov, Y. and Shakhlevich, N. (1995). Np-hardness of shop-scheduling problems with three jobs. *Discrete Applied Mathematics*, 59 :237–266. 29

Sotskov, Y., Tanaev, V., and Werner, F. (1976). Scheduling problems and mixed graph colorings. *Optimization*, 51(3) :597–624. 30

Storer, R. H., Wu, S. D., and Vaccari, R. (1992). New search spaces for sequencing problems with application to job shop scheduling. *Management science*, 38(10) :1495–1509. 68

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2) :278–285. 70

Timkovsky, V. (1998). Is a unit-time job shop not easier than identical parallel machines ? *Discrete Applied Mathematics*, 85 :149–162. 31

Xuong, H. (1992). Mathématiques discrètes et informatique. *Edition Masson*. 21

Yamada, T. and Nakano, R. (1992). A genetic algorithm applicable to large-scale job-shop problems. *In Parallel problem solving from nature*, pages 283–292. 68

**Résumé**

Ce travail porte sur l'étude de deux problèmes d'ordonnancement job shop avec des opérations de durée unitaire, où chacun de ces deux problèmes est réduit à un problème de coloration de graphes mixtes. Le premier s'intéresse au nombre chromatique et le second à la minimisation de la somme des couleurs des chemins de longueur maximale. Comme ces problèmes sont connus pour être NP-difficiles, des méthodes exactes et une métaheuristique sont proposées pour les résoudre. Pour chacun de ces problèmes de coloration de graphes, l'étude est réalisée en quatre étapes. D'abord une borne inférieure et une borne supérieure sont proposées pour le critère à optimiser. Ensuite, une modélisation basée sur la programmation linéaire à variables mixtes (MILP) indexant la couleur au niveau des variables décisionnelles est développée. Puis, deux variantes de l'algorithme par séparation et évaluation sont proposées, et finalement un algorithme de recherche tabou utilisant une structure dynamique de voisinage est adapté pour résoudre des instances de grande taille.

Des expérimentations menées sur des graphes mixtes, issus de plusieurs benchmarks modifiés et des instances générées aléatoirement, justifient l'efficacité et l'efficience des bornes et des méthodes de résolution proposées.

**Mots clés :** *coloration de graphes, ordonnancement, borne inférieure, MILP, séparation et évaluation, recherche tabou.*

**Abstract :**

This thesis deals with the study of two unit-time job shop scheduling problems, where each of these problems is reduced to a mixed graph coloring problem. The first problem deals with the chromatic number and the second addresses the sum of colors over all tails of maximal length paths. As these problems are known to be NP-hard, both exact and heuristic methods are proposed to solve them. For each of these graph coloring problems, the study is carried out in four steps. First, a new lower and upper bounds for the criteria to be optimized are given. Then, a color-indexed mathematical model using the proposed bounds is developed. Afterwards, two branch and bound algorithm variants are proposed, and finally a tabu search using a dynamic neighborhood structure is adapted for solving large instances.

Computational experiments conducted on several modified benchmarks show the efficiency and effectiveness of the proposed bounds and resolution methods.

**Keywords :** *graph coloring, scheduling, lower bound, MILP, branch and bound, tabu search*