

Université de Montréal

Nouveaux algorithmes, bornes et formulations pour les problèmes de la
clique maximum et de la coloration minimum

Par
Patrick St-Louis

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de Philosophiae Doctor (Ph.D.)
en informatique

Décembre, 2006

© Patrick St-Louis, 2006



QA

76

U54

2007

v. 016



Direction des bibliothèques

AVIS

L'auteur a autorisé l'Université de Montréal à reproduire et diffuser, en totalité ou en partie, par quelque moyen que ce soit et sur quelque support que ce soit, et exclusivement à des fins non lucratives d'enseignement et de recherche, des copies de ce mémoire ou de cette thèse.

L'auteur et les coauteurs le cas échéant conservent la propriété du droit d'auteur et des droits moraux qui protègent ce document. Ni la thèse ou le mémoire, ni des extraits substantiels de ce document, ne doivent être imprimés ou autrement reproduits sans l'autorisation de l'auteur.

Afin de se conformer à la Loi canadienne sur la protection des renseignements personnels, quelques formulaires secondaires, coordonnées ou signatures intégrées au texte ont pu être enlevés de ce document. Bien que cela ait pu affecter la pagination, il n'y a aucun contenu manquant.

NOTICE

The author of this thesis or dissertation has granted a nonexclusive license allowing Université de Montréal to reproduce and publish the document, in part or in whole, and in any format, solely for noncommercial educational and research purposes.

The author and co-authors if applicable retain copyright ownership and moral rights in this document. Neither the whole thesis or dissertation, nor substantial extracts from it, may be printed or otherwise reproduced without the author's permission.

In compliance with the Canadian Privacy Act some supporting forms, contact information or signatures may have been removed from the document. While this may affect the document page count, it does not represent any loss of content from the document.

Université de Montréal
Faculté des études supérieures

Cette thèse intitulée:

**Nouveaux algorithmes, bornes et formulations pour les problèmes de la
clique maximum et de la coloration minimum**

présentée par:

Patrick St-Louis

a été évaluée par un jury composé des personnes suivantes:

Jacques Ferland

président-rapporteur

Bernard Gendron

directeur de recherche

Alain Hertz

codirecteur

Michel Gendreau

membre du jury

Michael Trick

examineur externe

Ivo G. Rosenberg

représentant du doyen de la FES

Thèse acceptée le 25 janvier 2007

RÉSUMÉ

Nous présentons une méthode de décomposition qui permet d'améliorer toute fonction de borne supérieure sur la taille de la plus grande clique d'un graphe, et qui fournit comme sous-produit un sous-graphe induit ayant une grande probabilité de contenir la plus grande clique du graphe. Pour ce faire, notre méthode réduit progressivement la taille du graphe en éliminant les noeuds ayant la plus faible probabilité de faire partie d'une clique de taille maximum, et en mettant à jour une borne sur la taille de la plus grande clique.

Nous étudions aussi l'approche d'orientation des arêtes d'un graphe, afin d'obtenir une borne sur le nombre chromatique, dans le contexte de la recherche locale. Cette approche consiste à trouver l'orientation des arêtes du graphe de sorte que le plus long chemin dans le graphe orienté résultant soit le plus court possible. Le nombre de noeuds sur le plus long chemin d'une orientation optimale correspond au nombre chromatique du graphe. Nous étudions l'impact d'inverser, de retirer, ou d'ajouter un ou plusieurs arcs d'une orientation donnée sur la formation des circuits et des plus longs chemins du graphe, afin d'identifier des voisinages permettant d'éviter de créer des circuits (qui rendent difficile le calcul du plus long chemin) ainsi que des plus longs chemins (qui détériorent la qualité de la solution). Nous identifions et testons plusieurs stratégies en les incorporant dans un algorithme tabou, afin de donner une idée globale de l'efficacité de nos voisinages.

En écrivant le modèle mathématique du problème d'orientation des arêtes d'un graphe, et après plusieurs opérations sur ce modèle, nous obtenons une formulation pour le problème de la coloration par bande. Les transformations algébriques nous permettent de constater que les variables spécifiant les couleurs des noeuds sont fractionnaires, et seules les variables spécifiant l'orientation des arêtes sont binaires. Le nombre de ces variables est égal au nombre d'arêtes dans le graphe, ce qui en fait une formulation compacte pour la coloration par bande (et par le fait même, pour la coloration standard qui est un cas particulier). L'intérêt pour une telle formulation augmente avec les poids sur les arêtes puisque seuls quelques coefficients changent,

contrairement aux approches usuelles où le nombre de variables dépend du nombre chromatique et donc des valeurs sur les arêtes. Notre formulation utilise une borne supérieure sur le nombre chromatique pour calculer ses coefficients, tout comme un modèle standard utiliserait une borne supérieure sur le nombre chromatique pour définir ses variables. Dans les cas où calculer une telle borne serait difficile, nous introduisons une formulation normalisée pour laquelle une telle valeur n'est pas nécessaire.

Mots clés : Décomposition, Orientation d'arêtes, Inversion d'arêtes, Recherche locale, Tabou, Coloration par bande.

ABSTRACT

We present a decomposition method which improves any upper bound function on the size of a maximum clique of a graph, and gives as a by-product an induced subgraph having a good probability of containing a maximum clique of the graph. To do so, our method progressively reduces the size of the graph by eliminating vertices that have the least probability of being a part of a maximum clique, and by updating a bound on the size of the largest clique.

We also study the edge-orientation approach to obtain a bound on the chromatic number in a local search context. This approach consists in finding the edge-orientation which minimises the length of the longest path. The number of vertices on a longest path of an optimal edge-orientation is equal to the chromatic number. We study the impact of the reversal, removal, or insertion of one or more arcs of a given edge-orientation on the formation of circuits and on the longest paths, in order to identify neighborhoods that allow to avoid the creation of such circuits (which make the computing of the longest path more difficult) and to avoid creating even longer paths (which deteriorate the quality of the solution). We identify and test many strategies by incorporating them in a tabu search algorithm, in order to give a global idea of the efficiency of our neighborhoods.

When writing the mathematical model of the edge-orientation problem, and after multiple operations on the model, we obtain a formulation for the bandwidth coloring problem. This algebraic work allows us to notice that variables specifying the vertices' colors are fractional, and only the variables defining the orientation of the edges are binary. The number of such variables is equal to the number of edges in the graph, which makes it a compact formulation for the bandwidth coloring problem (and in turn, for the standard coloring problem, which is a special case). The interest for such a formulation increases with the weights on the edges since only a few coefficients change in our formulation, opposed to other formulations where the number of variables depend on the chromatic number which in turn depends on the weights on the edges. Our formulation uses an upper bound on

the chromatic number to compute coefficients, as other formulations would use the chromatic number to define its variables. In cases where computing an upper bound would be difficult, we introduce a normalized formulation for which it is not necessary to compute a bound.

Keywords: Decomposition, Edge orientation, Edge reversal, Local search, Tabu, Bandwidth coloring.

TABLE DES MATIÈRES

RÉSUMÉ	iii
ABSTRACT	v
TABLE DES MATIÈRES	vii
LISTE DES TABLEAUX	x
LISTE DES FIGURES	xi
DÉDICACE	xii
REMERCIEMENTS	xiii
INTRODUCTION	1
CHAPITRE 1 : PROBLÉMATIQUES DE LA CLIQUE MAXIMUM ET DE LA COLORATION MINIMUM	4
1.1 Terminologie	4
1.2 Complexité	6
1.3 Modèles mathématiques	8
1.4 Méthodes de résolution	9
1.4.1 Méthodes exactes	10
1.4.2 Méthodes heuristiques (métaheuristiques)	13
1.4.3 Familles de graphes	18
1.4.4 Programmation non linéaire	18
1.4.5 Programmation semi-définie	19
1.4.6 Méthodes d'approximation	20
1.5 Revue de littérature	20
1.5.1 Problème de la clique maximum	20
1.5.2 Problème de la coloration minimum	23

1.5.3	Problème de la coloration par bande et autres généralisations	27
-------	---	----

CHAPITRE 2 : MÉTHODE DE DÉCOMPOSITION POUR LE PROBLÈME DE LA CLIQUE MAXIMUM 30

2.1	Méthode de décomposition	30
2.2	Extraction d'un sous-graphe pour le calcul d'une borne inférieure .	33
2.3	Introduction	36
2.4	The Decomposition Algorithm	37
2.5	Using the Decomposition Algorithm to Compute Lower Bounds . .	42
2.6	Computational Results	45
2.6.1	Computing Upper Bounds	45
2.6.2	Computing Lower Bounds	49
2.7	Conclusion	53

CHAPITRE 3 : ÉTUDE DE VOISINAGES BASÉS SUR L'ORIENTATION D'ARÊTES POUR LE PROBLÈME DE LA COLORATION MINIMUM 55

3.1	Voisinages	56
3.1.1	Inversion d'un arc	56
3.1.2	Inversion de plusieurs arcs reliés à un noeud	57
3.1.3	Inversion de plusieurs arcs parmi des composantes connexes	57
3.1.4	Ajout et retrait d'arcs	58
3.2	Résultats expérimentaux	59
3.3	Introduction	61
3.4	Properties	65
3.5	Four tabu search algorithms	77
3.6	Computational experiments and conclusions	81

CHAPITRE 4 : MODÈLES DE PROGRAMMATION MATHÉMATIQUE POUR LE PROBLÈME DE LA COLORATION PAR BANDE 85


4.1	Orientation d'arêtes et coloration par bande	85
4.2	La formulation en nombres entiers	94
4.3	La formulation en nombres entiers normalisée	98
4.4	Résultats préliminaires	100
4.5	Conclusion	105
CONCLUSION		106
BIBLIOGRAPHIE		109

LISTE DES TABLEAUX


2.1	Upper bounds obtained with five upper bound functions $h_x(G)$. . .	47
2.2	Upper bounds obtained with five upper bound functions $h_x(G)$. . .	48
2.3	Average improvement for each family of graphs	49
2.4	Lower bounds $l_x(G)$ and $l_x(G^*)$	51
2.5	Lower bounds $l_x(G)$ and $l_x(G^*)$	52
3.1	Descent algorithms on ten random graphs.	82
3.2	Tabu search algorithms on ten random graphs.	82
3.3	Statistics on random graphs.	82
3.4	Tabu search algorithms on DIMACS benchmark graphs.	83
4.1	Résultats complets pour la coloration par bande	102
4.2	Résultats complets pour la coloration par bande (suite)	103
4.3	Nombre d'instances où l'optimalité est prouvée (sur 33)	103
4.4	Nombre d'instances où la réalisabilité est atteinte (sur 33)	104

LISTE DES FIGURES

1.1	Graphes non orienté (G_1), orienté (G_2), et mixte (G_3)	4
1.2	Graphe numéroté (G_1^a), coloré (G_1^b), et coloré avec numéros (G_1^c)	5
3.1	Illustration of a notation	62
3.2	A sequence of neighbor solutions	69
3.3	Illustration of the <i>total orienting strategy</i> with neighborhood N_3	73
3.4	Basic tabu search	77
4.1	La différence d'amplitude est toujours inférieure à $\chi_a(G) - 1 = 4$	98



À ma femme, qui a su trouver de la patience
là où personne d'autre n'en aurait trouvé.



REMERCIEMENTS

Mes profonds remerciements vont à monsieur Bernard Gendron pour avoir eu confiance en mes idées et ma capacité de me rendre jusqu'au bout, tout en étant présent le long de ce trajet afin de me guider dans les passages les plus ardues. Ses enseignements m'ont servi tout au long de mes études supérieures, et il a été une grande inspiration par son ardeur au travail. Cette thèse est le résultat d'une progression difficile au niveau de la rédaction, et ses nombreux conseils m'ont permis de produire un texte de bonne qualité. Enfin, sa jovialité a réussi à me rassurer dans les moments d'incertitude, et m'a permis de poursuivre mes études dans un climat des plus agréables.

Je remercie monsieur Alain Hertz d'abord pour m'avoir inspiré l'envie de travailler sur les graphes au tout début de mes études, puis pour m'avoir aidé à boucler le cycle de mes études supérieures dans un domaine des plus captivants. Sa capacité à correctement identifier la source d'un problème ou le cas particulier pour lequel une affirmation ne tient pas, en fait un analyste hors pair et un correcteur des plus intransigeants. Toutefois, la satisfaction d'un travail bien fait n'en est que plus grande lorsque l'on obtient son approbation. Enfin, son humour et ses anecdotes ont agrémenté toutes nos rencontres, et c'est avec regret (un peu mitigé tout de même) que je termine cette thèse, tout en conservant l'espoir de collaborations futures.

Je tiens à remercier madame Mariette Paradis pour sa patience dans la gestion de mon dossier administratif, et pour m'avoir éclairé lorsque je me trouvais dans la totale ignorance.

Je remercie tout particulièrement ma femme pour m'avoir supporté dans mes décisions et tout au long de mon travail à la maison, mes beaux-parents pour avoir donné une quantité incroyable de leur temps, mes parents pour leur soutien

moral et financier, et mes collègues de maîtrise pour leur amitié, en particulier Francis, mon souffre-douleur.

Enfin, je remercie l'Université de Montréal, le Département d'informatique et de recherche opérationnelle, et l'association étudiante pour m'avoir fourni un environnement stimulant pour la recherche et le développement académique.

INTRODUCTION

La théorie des graphes est un domaine commun des mathématiques et de la science informatique qui prend ses racines principalement au 19e siècle, lors de l'introduction du problème des quatres couleurs. Ce problème consiste à déterminer s'il est possible, avec seulement quatre couleurs, de colorer chaque pays d'une carte géographique de sorte que deux pays partageant une frontière n'aient pas la même couleur. Ce problème fut l'événement instigateur de nombreux concepts qui constituèrent ce qu'on appela plus tard la théorie des graphes.

On définit un graphe comme un ensemble de noeuds et un ensemble d'arêtes, chaque arête reliant deux noeuds. Le graphe associé au problème des quatres couleurs est formé en créant un noeud pour chaque pays et en reliant par une arête toute paire de noeuds dont les pays correspondants ont une frontière commune. Le problème des quatres couleurs revient donc à déterminer s'il est possible, avec seulement quatre couleurs, d'en affecter une à chaque noeud du graphe, de sorte que deux noeuds qui sont reliés par une arête n'aient pas la même couleur.

Ce problème a été généralisé comme suit : étant donné un graphe, combien de couleurs sont nécessaires pour colorer tous ses noeuds de telle façon que deux noeuds reliés par une arête n'aient pas la même couleur ? Le nombre minimum de couleurs requises est appelé le nombre chromatique. Trouver le nombre chromatique d'un graphe est un problème classique, ayant été étudié depuis les débuts de la théorie des graphes [Bro41].

De nombreux problèmes provenant de la théorie des graphes sont difficiles à résoudre, car il arrive souvent que la seule façon connue de les résoudre est de tester une grande partie des combinaisons possibles. C'est un peu comme tenter d'ouvrir un coffre-fort sans en connaître la combinaison ; il n'existe pas vraiment d'autres façons que de tester toutes les combinaisons possibles. En général, la structure du graphe nous permet d'éviter de tester un bon nombre de combinaisons, mais cela suffit rarement à simplifier le problème suffisamment pour permettre de le résoudre de manière efficace. De plus, le nombre de combinaisons à tester croît

exponentiellement avec le nombre de noeuds dans le graphe, ce qui implique qu'en pratique, on ne peut résoudre à l'optimum que les problèmes définis sur des graphes de petite taille.

Le problème de **coloration de graphes**, qui consiste à trouver le nombre chromatique d'un graphe, est l'un des sujets traités dans cette thèse. En particulier, nous étudions le problème de trouver l'orientation des arêtes du graphe qui minimise la longueur du plus long chemin dans le graphe, un problème équivalent à colorer les noeuds du graphe. Plus particulièrement, nous nous intéressons à ce problème dans le cadre du développement d'une méthode de recherche locale. Nous introduisons aussi deux formulations pour la **coloration par bande**, une généralisation du problème de coloration de graphes, où des poids sur les arêtes restreignent davantage les choix de couleurs possibles pour colorer les noeuds. Ces formulations, basées sur les orientations des arêtes du graphe, sont compactes, car leur taille dépend uniquement de l'existence des arêtes et non pas de leur poids. Enfin, nous étudions le problème de la **clique maximum**, qui consiste à trouver la taille de la plus grande clique d'un graphe. Une clique est un sous-graphe complet, c'est-à-dire un ensemble de noeuds qui sont tous reliés deux à deux par une arête. Notons que la taille de la plus grande clique est une valeur fréquemment utilisée pour borner inférieurement le nombre chromatique (l'inverse est aussi vrai : le nombre chromatique est une valeur fréquemment utilisée pour borner supérieurement la taille de la plus grande clique). Nous présentons une méthode de décomposition qui permet d'améliorer toute méthode produisant une borne supérieure sur la taille de la plus grande clique d'un graphe. En particulier, nous améliorons d'environ 60% les bornes sur la taille de la plus grande clique produites par des heuristiques de coloration de graphes.

Le premier chapitre présente une introduction à la terminologie, à la complexité, aux méthodes de résolution, ainsi qu'une revue de la littérature concernant les trois problèmes traités dans cet ouvrage. Les trois chapitres suivants introduisent respectivement les approches de résolution pour la clique maximum, la coloration par orientation d'arêtes et la coloration par bande, ainsi que les articles associés.

Avant chaque article (rédigé en anglais), nous décrivons les grandes lignes de son contenu, en ajoutant parfois quelques informations supplémentaires qui n'entrent pas dans le cadre de la publication. Chaque chapitre sert entre autres à expliquer les idées de l'article dans un contexte plus large, tout en référant le lecteur à l'article lui-même lorsqu'il s'agit de preuves et démonstrations. La conclusion se trouve dans le dernier chapitre, ainsi qu'une discussion sur les pistes de recherche découlant de cette thèse.

Les articles ont été rédigés en collaboration avec Bernard Gendron et Alain Hertz. L'auteur de la présente thèse a contribué à l'ensemble des idées et la majorité des preuves, la totalité de la programmation informatique et des tests pratiques, ainsi que le premier jet des articles. La contribution des coauteurs inclut le peaufinage des idées, leur généralisation dans certains cas, l'élaboration de plusieurs preuves et une grande part de la rédaction. Le premier article, intitulé "A Decomposition Algorithm for Computing Bounds on the Clique Number of a Graph", a été soumis pour publication à "Discrete Optimization". Le second article, intitulé "On Edge Orienting Methods for Graph Coloring", a été accepté pour publication dans "Journal of Combinatorial Optimization". Le troisième article, intitulé "A Graph Orientation Formulation for Bandwidth Coloring", est en cours de rédaction.

CHAPITRE 1

PROBLÉMATIQUES DE LA CLIQUE MAXIMUM ET DE LA COLORATION MINIMUM

1.1 Terminologie

Les problèmes d'horaires, d'ordonnancement de tâches, d'affectation de ressources et de gestion de conflits sont tous des exemples de situations qui peuvent être représentées par un graphe G , constitué de noeuds $V(G)$ (ou sommets) et d'arêtes $E(G)$ ou d'arcs $A(G)$ entre certaines paires de noeuds. Lorsqu'il n'y a aucune ambiguïté possible, on peut les dénoter simplement V , E et A . À chaque noeud du graphe est associée une entité du problème (par exemple, une ressource), et une arête (ou un arc) relie deux noeuds qui sont en conflit (ou ont un lien quelconque entre eux). Tous les graphes qui sont traités dans cette thèse sont simples, c'est-à-dire que tous les arcs et arêtes sont distincts et relient des noeuds distincts. Un graphe est orienté s'il n'a que des arcs, et il est non orienté s'il ne possède que des arêtes. Les graphes possédant à la fois des arcs et des arêtes sont appelés graphes mixtes, et seront exclus de cette thèse. Quelques exemples sont illustrés dans la Figure 1.1.

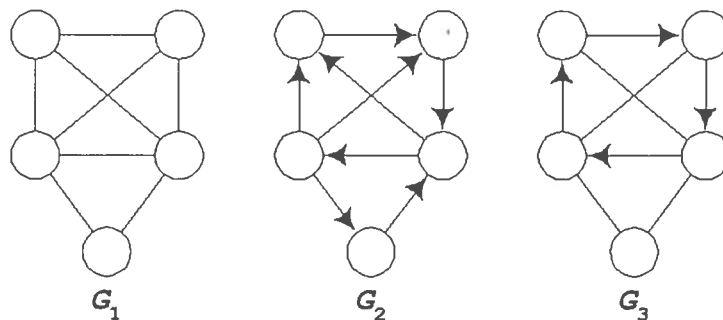


FIG. 1.1 – Graphes non orienté (G_1), orienté (G_2), et mixte (G_3)

Un sous-graphe $G' \subseteq G$ est un graphe tel que $V(G') \subseteq V(G)$, $E(G') \subseteq E(G)$ et $A(G') \subseteq A(G)$. Un sous-graphe $G' \subseteq G$ est induit si toutes les arêtes (ou arcs) de G qui relient deux noeuds de G' font partie de G' . Un graphe non orienté est dit complet s'il existe une arête entre chaque paire de noeuds du graphe. Une clique de G est un sous-graphe induit complet de G . De façon similaire, un stable (ou ensemble indépendant) est un sous-graphe induit qui ne possède aucune arête. Une coloration des noeuds d'un graphe non orienté G est une affectation d'une couleur à chaque noeud, de sorte que deux noeuds reliés par une arête n'ont pas la même couleur. Les noeuds d'une clique ont ainsi tous des couleurs différentes. On représente généralement une couleur par un nombre non négatif, et c'est ce nombre qui est utilisé lorsqu'on parle de la différence entre deux couleurs, qui signifie la valeur absolue de la différence entre les nombres associés à ces deux couleurs.

En général, pour identifier les noeuds d'un graphe, on donne à chacun un numéro distinct (voir le graphe G_1^a de la Figure 1.2). Il est ainsi plus facile d'identifier un sous-ensemble de noeuds, par exemple une clique de taille 4 ($\{1, 2, 3, 4\}$). Le graphe suivant (G_1^b) dans la même figure a été coloré en utilisant quatre couleurs. Tel que mentionné précédemment, on attribue aussi un nombre à chaque couleur, comme c'est le cas dans le graphe G_1^c , ce qui permet plus facilement leur manipulation dans des algorithmes et des modèles mathématiques.

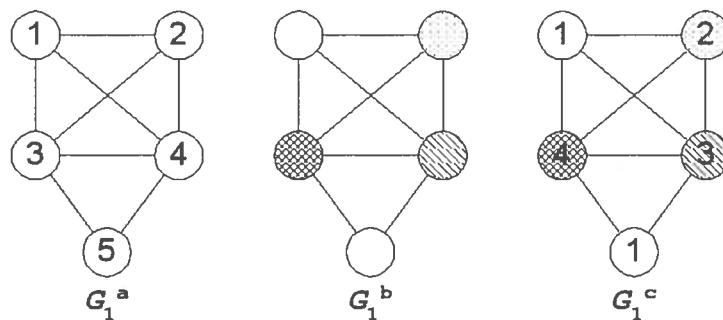


FIG. 1.2 – Graphe numéroté (G_1^a), coloré (G_1^b), et coloré avec numéros (G_1^c)

1.2 Complexité

Un problème est catégorisé dans la classe NP si c'est un problème de décision (sa réponse est oui ou non) et s'il peut se résoudre à l'aide d'un algorithme non-déterministe polynomial. Un algorithme non-déterministe consiste, dans le cas de problèmes de décision, à choisir itérativement un élément parmi une liste de choix et où la séquence des choix effectués peut soit être acceptée, soit être refusée, ou bien l'algorithme peut boucler indéfiniment. Les choix sont effectués à l'aide d'un oracle qui devine quel est le bon choix à effectuer à chaque itération. Cette notion d'oracle est importante car un algorithme non-déterministe ne spécifie pas comment faire un choix, mais seulement quoi faire une fois qu'un choix a été fait. Par exemple, un algorithme non-déterministe qui résout le problème de savoir s'il existe un chemin de longueur k ($2 \leq k \leq |V|$) dans un graphe consiste à choisir un premier noeud, puis un noeud à chaque itération subséquente et refuser la séquence si le noeud choisi n'est pas un successeur du noeud précédent et accepter la séquence si sa taille atteint k . L'algorithme ne peut pas boucler indéfiniment car il fera au plus k itérations et k est un nombre fini. Si l'algorithme refuse la séquence, c'est qu'il n'existe pas de chemin de longueur k , sinon l'oracle aurait fait les choix permettant de l'identifier correctement.

Le temps requis par un algorithme non-déterministe est calculé en déterminant le temps minimum requis pour accepter une séquence de choix. On considère qu'effectuer un choix (en utilisant l'oracle) prend un temps constant, c'est-à-dire un temps qui ne dépend pas de la taille de l'instance du problème (dans cette thèse, une instance correspond à un graphe). Si le temps total requis par un algorithme non-déterministe est proportionnel à un polynôme en fonction de la taille de l'instance du problème, alors cet algorithme est dit polynomial. Dans notre exemple, ce temps serait en pire cas de $|V|$ itérations multiplié par le temps que prend une itération, c'est-à-dire effectuer un choix et vérifier si le noeud choisi est un successeur du noeud précédent. Cette opération implique (en pire cas) de parcourir au complet la liste des arêtes ($|E|$) ou des arcs ($|A|$) du graphe. Donc, le temps

requis par notre algorithme est proportionnel à $|V| * |E|$ ou $|V| * |A|$, ce qui est un polynôme en fonction de la taille de l'instance. On utilise le terme “dans l'ordre de” pour spécifier la complexité en pire cas d'un algorithme ; celui décrit ci-haut est donc dans l'ordre de $|V| * |E|$ ou $|V| * |A|$.

Un algorithme non-déterministe n'est pas utilisable en pratique pour résoudre un problème NP, puisqu'il ne spécifie pas comment effectuer un choix. Choisir au hasard n'est pas une option efficace car la probabilité d'accepter une séquence dans un tel cas est infime. En fait, il n'existe aucune façon connue d'effectuer un choix efficacement pour toutes les instances possibles. L'une des raisons qui rend ces problèmes intéressants est qu'il n'existe aussi aucune preuve de l'absence de façons de choisir efficacement pour toute instance. En d'autres mots, la notion d'algorithme non-déterministe nous permet de supposer qu'il existe une façon efficace d'effectuer les choix, mais ne nous l'assure pas.

Les problèmes de déterminer si un graphe possède une clique de taille k , ou de déterminer s'il existe une façon de colorer le graphe en utilisant k couleurs, font partie de la classe de complexité NP-complet (incluse dans NP). Un problème fait partie de cette classe si toute instance d'un problème NP peut se transformer en instance pour ce problème en temps polynomial en fonction de la taille de l'instance. Ainsi, s'il était possible de résoudre efficacement (en temps polynomial) un problème NP-complet, alors il serait possible de résoudre efficacement tous les problèmes NP, car la complexité de la transformation puis de la résolution reste polynomial en fonction de la taille de l'instance.

Les problèmes de trouver la taille de la plus grande clique ou le nombre chromatique d'un graphe sont dans la classe NP-difficile. Un problème fait partie de cette classe si ce n'est pas un problème de décision, mais que le résoudre efficacement permettrait aussi de résoudre efficacement un problème NP. Par exemple, si l'on possédait un algorithme efficace pour trouver la taille de la plus grande clique d'un graphe, alors pour toute instance du problème de déterminer s'il existe une clique de taille k , on n'aurait qu'à trouver la taille de la plus grande clique, et répondre “vrai” si k est inférieur ou égal à la taille de la plus grande clique, et “faux” sinon.

Une autre façon de voir la relation est que si l'on possédait un algorithme efficace pour résoudre un problème NP-difficile, alors on pourrait l'utiliser pour effectuer les choix dans un algorithme non-déterministe polynomial, et ainsi résoudre en temps polynomial un problème NP.

1.3 Modèles mathématiques

Pour décrire un problème d'optimisation, on écrit généralement un modèle mathématique représentant son objectif et ses contraintes sous formes d'équations et d'inéquations. Un modèle est linéaire si son objectif et ses contraintes sont linéaires, et il est non linéaire si ce n'est pas le cas. Si certaines variables sont restreintes à prendre des valeurs entières ou binaires (0 ou 1), et que le modèle est linéaire, alors c'est un modèle de programmation linéaire en nombres entiers.

Il existe souvent plusieurs façons de modéliser un problème, mais les modèles résultants ne sont pas toujours équivalents. Pour comparer deux modèles de programmation linéaire en nombres entiers, par exemple, on relaxe la contrainte d'intégralité des variables entières, puis on compare les domaines réalisables issus de cette relaxation. On préférera généralement le modèle ayant le plus petit domaine réalisable, car le temps de recherche en est souvent réduit, mais il arrive que certains modèles soient plus faciles à utiliser ou soient plus propices à la génération de coupes (des contraintes supplémentaires qui restreignent le domaine réalisable).

Un modèle mathématique standard de programmation linéaire en nombres entiers associé au problème de trouver la plus grande clique d'un graphe est le suivant. On introduit une variable binaire x_i à chaque noeud, valant 1 si le noeud i est sélectionné pour faire partie de la plus grande clique, et 0 sinon. On cherche donc à maximiser la somme des x_i , tout en s'assurant que deux noeuds qui ne sont pas reliés par une arête ne puissent être sélectionnés en même temps. Ceci est obtenu en vérifiant que la somme de toute paire de variable $x_i + x_j$ soit inférieure ou égale à 1 s'il n'y a pas d'arête entre i et j .

$$\begin{aligned}
& \text{Max} \quad \sum_{i \in V} x_i, \\
& \text{s.à} \quad x_i + x_j \leq 1 \quad (i, j) \notin E, \\
& \quad \quad x_i \in \{0, 1\} \quad i \in V.
\end{aligned}$$

Un modèle mathématique standard de programmation linéaire en nombres entiers associé au problème de trouver le nombre chromatique d'un graphe est le suivant. On introduit une variable binaire x_{ik} à chaque noeud i et couleur k , valant 1 si le noeud i est coloré en utilisant la couleur k , et 0 sinon. On introduit aussi une variable binaire w_k pour chaque couleur, valant 1 si la couleur k est utilisée, et 0 sinon. On cherche donc à minimiser la somme des w_k , tout en s'assurant qu'un noeud est coloré d'une seule couleur, que sa couleur est accessible (en la comparant à w_k), et en vérifiant que toute paire de noeuds reliés par une arête ne partagent pas la même couleur. Enfin, on dénotera par $\bar{\chi}$ l'ensemble des couleurs disponibles pour colorer les noeuds du graphe.

$$\begin{aligned}
& \text{Min} \quad \sum_{k \in \bar{\chi}} w_k, \\
& \text{s.à} \quad \sum_{k \in \bar{\chi}} x_{ik} = 1 \quad i \in V, \\
& \quad \quad x_{ik} \leq w_k \quad i \in V, k \in \bar{\chi}, \\
& \quad \quad x_{ik} + x_{jk} \leq 1 \quad (i, j) \in E, k \in \bar{\chi}, \\
& \quad \quad x_{ik} \in \{0, 1\} \quad i \in V, k \in \bar{\chi}, \\
& \quad \quad w_k \in \{0, 1\} \quad k \in \bar{\chi}.
\end{aligned}$$

1.4 Méthodes de résolution

Pour résoudre un problème NP-difficile, les stratégies utilisées se regroupent principalement en six catégories, que nous décrivons ci-après. La section suivante présentera la littérature sur ces méthodes appliquées aux problèmes de la clique maximum et de la coloration minimum.

1.4.1 Méthodes exactes

Cette catégorie contient tous les algorithmes qui, lorsqu'ils se terminent, prouvent l'optimalité de la solution qu'ils produisent. De tels algorithmes ne peuvent généralement être utilisés que sur des instances de petite taille, car le temps et la puissance de calcul requis pour la complétion de l'algorithme croissent rapidement en fonction de la taille de l'instance que l'on tente de résoudre. La plupart des algorithmes de cette catégorie sont basés sur la philosophie "diviser-pour-régner", qui prétend qu'on peut simplifier la résolution d'un problème en le divisant en sous-problèmes, puisqu'un sous-problème est en général plus facile à résoudre que le problème original.

Pour plus d'informations sur les méthodes exactes et la programmation linéaire en nombres entiers, voir [Wol98].

1.4.1.1 Génération de colonnes

La génération de colonnes est une méthode pour résoudre des problèmes linéaires ayant un grand nombre de variables. Au lieu de résoudre le problème pour toutes les variables en même temps, on ne fait que le résoudre pour un petit nombre de variables, puis on ajoute itérativement des variables permettant d'améliorer la valeur de la solution. Lorsqu'aucune variable ne peut être ajoutée tout en améliorant la valeur de la solution, l'algorithme se termine. Il existe donc un problème "principal", contenant un sous-ensemble de variables du problème original, et un problème "secondaire" qui, lorsqu'on le résout, identifie une variable à insérer dans le problème "principal" (ou indique qu'il n'y a pas de telles variables).

1.4.1.2 Énumération (force brute)

Un algorithme d'énumération consiste à énumérer toutes les solutions réalisables, pour ne conserver que la meilleure. Cette méthode, lorsqu'elle se termine, prouve l'optimalité simplement en démontrant qu'il ne peut exister de solution meilleure que celle trouvée car elle aurait nécessairement été énumérée par l'algo-

rithme. On peut améliorer un algorithme d'énumération en utilisant des stratégies permettant d'identifier (sans les énumérer) des sous-ensembles de solutions dont la valeur n'est pas optimale. On évite ainsi le travail de les énumérer.

1.4.1.3 Branch-and-bound

Un algorithme de branch-and-bound consiste à partitionner itérativement le domaine réalisable en sous-problèmes de plus petite taille, qui sont en général plus faciles à résoudre, jusqu'à ce que l'on puisse trouver leur solution optimale de manière efficace (généralement lorsque le sous-problème est trivial à résoudre, par exemple pour le problème de la clique maximum, un sous-problème est trivial si c'est un graphe sans noeuds ni arêtes, ou une clique, ou un stable). L'étape consistant à diviser un problème en sous-problèmes est appelé un "branchement" (branch). Lorsqu'un sous-problème est résolu à l'optimum, on peut en extraire une solution réalisable pour le problème original en observant les différents branchements effectués avant d'arriver au sous-problème en question. L'algorithme garde en mémoire la meilleure solution réalisable obtenue dans les itérations précédentes afin de réduire l'effort de calcul pour les branchements subséquents. En effet, après avoir effectué un branchement, l'algorithme évalue le potentiel de chaque sous-problème (branche) en bornant sa valeur optimale à l'aide d'une méthode heuristique ou en résolvant une relaxation du sous-problème. Si la borne est de qualité moindre ou égale à la valeur de la meilleure solution réalisable obtenue préalablement, alors la branche est coupée, c'est-à-dire que l'on n'effectuera pas de branchement sur ce sous-problème. L'étape consistant à évaluer le potentiel d'un sous-problème est appelé "évaluation" (bound), d'où l'appellation branch-and-bound. Cette méthode est exacte puisque la solution optimale fait partie d'au moins un sous-problème, et que cette branche ne sera jamais coupée (sauf si une solution optimale différente a été trouvée préalablement).

1.4.1.4 Branch-and-cut

Dans le cadre de la programmation linéaire en nombres entiers, on utilise souvent une méthode de coupes pour produire des inégalités valides (contraintes) qui ne restreignent pas le problème en nombres entiers, mais plutôt le problème où les contraintes d'intégralité des variables entières sont relaxées. Un algorithme de branch-and-cut consiste à utiliser une méthode de coupes à l'intérieur d'un algorithme de branch-and-bound lors de l'évaluation d'une branche. En effet, si l'on résout à l'optimum le sous-problème linéaire obtenu en relaxant les contraintes d'intégralité des variables entières, on obtient une évaluation de la qualité de la branche. En utilisant une méthode de coupes, on peut améliorer cette évaluation en rendant la solution optimale fractionnaire non réalisable pour le problème relaxé. Les inégalités valides ainsi produites sont appelées "locales" si elles ne peuvent être appliquées qu'au sous-problème, et elles sont appelées "globales" si elles sont valides pour le problème original.

1.4.1.5 Branch-and-price

Similaire à l'algorithme de branch-and-cut, l'algorithme de branch-and-price ajoute des variables au lieu d'ajouter des contraintes, en appliquant un algorithme de génération de colonnes lors de l'évaluation des branches. Cette approche est particulièrement efficace pour les problèmes de grande taille, qui sont difficiles à résoudre avec un algorithme de branch-and-bound ou de branch-and-cut.

1.4.1.6 Programmation par contraintes

La programmation par contraintes sert principalement à résoudre des problèmes de satisfaction de contraintes, i.e., on cherche s'il existe une solution qui satisfait toutes les contraintes. Il n'y a donc aucun objectif particulier, seulement des contraintes et un espace de solutions. Toutefois, la plupart des problèmes NP sont définis uniquement par leurs contraintes, ce qui fait qu'ils se prêtent bien à la résolution par cette approche. Les outils de base utilisés par la programmation par

contraintes proviennent principalement de la programmation logique, son ancêtre en quelque sorte. *Prolog* (programmation logique) est l'un des premiers langages de programmation dans ce domaine.

1.4.2 Méthodes heuristiques (métaheuristiques)

Cette catégorie contient tous les algorithmes qui tentent de trouver une solution optimale ou simplement sa valeur, sans toutefois prouver l'optimalité du résultat lorsqu'ils se terminent. En général, ces méthodes fournissent une solution de bonne qualité, ou une borne sur la valeur optimale. Contrairement aux méthodes exactes, des méthodes heuristiques peuvent être utilisées pour résoudre des instances de très grande taille. On peut conjuguer deux méthodes heuristiques, l'une produisant une borne inférieure et l'autre une borne supérieure sur la valeur optimale, afin de produire un intervalle à l'intérieur duquel se trouve la valeur optimale. Si l'intervalle ne contient qu'une seule valeur, alors l'optimalité de cette valeur est prouvée. Fréquemment, une métaheuristique doit se choisir un point de départ à partir duquel l'algorithme progresse itérativement ; on peut donc répéter l'exécution d'une métaheuristique en utilisant plusieurs points de départ et ainsi obtenir des résultats fort différents.

Pour plus d'informations sur les métaheuristiques, nous référons au livre de Glover et Kochenberger [GK03].

1.4.2.1 Énumération partielle

On appelle énumération partielle tout algorithme exact (tel que défini dans la section précédente) dont on interrompt l'exécution avant la fin en limitant le temps d'exécution ou le nombre d'itérations, par exemple. On obtient généralement une borne sur la valeur optimale, et parfois un intervalle.

1.4.2.2 Glouton

Toute solution (même optimale) peut être décrite à l'aide d'une suite de décisions non-déterministes (voir Section 1.2). Toutefois, en pratique, un algorithme se doit de prendre des décisions déterministes. Un glouton est un algorithme qui prend itérativement une telle décision, sans jamais la remettre en question dans les itérations subséquentes, afin de définir progressivement une solution réalisable pour le problème étudié. En général, la solution obtenue n'est pas optimale, car il est difficile de déterminer avec assurance quelles décisions prendre pour atteindre une solution optimale. Pour le problème de la clique maximum, l'algorithme qui choisit un premier noeud, puis à chaque itération subséquente choisit un autre noeud qui est connecté à tous les noeuds choisis préalablement, jusqu'à ce qu'il ne reste plus aucun noeud qui satisfasse cette condition, est un algorithme glouton.

1.4.2.3 Recherche locale, réactive, adaptative

La recherche locale consiste premièrement à définir la notion de "voisinage d'une solution", une fonction qui, à partir d'une solution donnée, détermine un ensemble de solutions "voisines" de celle-ci. Pour décrire cette fonction, on utilise généralement la notion de mouvement qui consiste en un ensemble d'opérations permettant de transformer une solution donnée en une autre solution. Ensuite, l'algorithme démarre à partir d'une première solution réalisable et explore son voisinage à la recherche d'une solution de meilleure valeur (en général). Lorsque son choix est fait, alors l'algorithme utilise la nouvelle solution pour générer un nouveau voisinage. L'algorithme progresse ainsi itérativement d'une solution à une autre en tentant de trouver une suite de solutions menant à une solution optimale. L'efficacité d'une telle méthode est reliée à la définition des voisinages ainsi qu'à l'efficacité de la recherche à l'intérieur des voisinages. D'autres facteurs entrent aussi en ligne de compte, comme la capacité de sortir d'un optimum local (solution où tout mouvement permis mène à une solution de moindre qualité), ou les critères d'arrêt (limites sur le nombre total d'itérations ou le nombre depuis la dernière

amélioration, le temps de recherche, la qualité du voisinage, etc.).

Un algorithme de descente est une recherche locale ne permettant pas de choisir dans le voisinage une solution de qualité moindre que la solution courante. Une descente s'arrête généralement lorsque toutes les solutions du voisinage sont de qualité moindre que la solution courante. On la dit stricte si l'algorithme s'arrête lorsque toutes les solutions du voisinage sont de qualité moindre ou égale à la solution courante.

Tout algorithme de recherche locale qui n'est pas une descente permet donc de choisir une solution dans le voisinage qui est de qualité moindre que la solution courante. Or, ceci peut provoquer un va-et-vient entre deux (ou plusieurs) solutions, car en général, si une solution "b" fait partie du voisinage d'une solution "a", alors "a" fait aussi partie du voisinage de la solution "b". Donc, lorsque l'algorithme explore le voisinage de la solution "b" de qualité moindre que la solution "a", il est possible que l'algorithme choisisse la solution "a" et ainsi revienne sur ses pas. Les algorithmes de recherche locale (autre que la descente) se distinguent donc principalement par leur mécanisme pour éviter de retourner sur leurs pas.

Un algorithme tabou est une recherche locale qui conserve une liste d'attributs "tabous", et où tout mouvement possédant un attribut tabou est interdit (sauf s'il mène à une solution meilleure que toutes celles rencontrées au préalable). Deux algorithmes tabous se distinguent principalement par leur gestion de la liste tabou (sa longueur, son contenu, sa mise à jour, etc.). En général, une fois qu'un mouvement est effectué, on ajoute dans la liste un attribut interdisant le mouvement inverse. Toutefois, la notion d'attribut permet d'interdire plus de mouvements que simplement le mouvement inverse. Par exemple, on pourrait interdire tout mouvement qui mène à la solution précédente ou à une solution semblable.

Un recuit simulé est une recherche locale qui, à chaque itération, choisit au hasard une solution s' dans le voisinage de la solution courante s , puis détermine une probabilité d'accepter cette solution en fonction de la qualité de celle-ci et d'un facteur T appelé "température". Au départ, la température est initialisée à une grande valeur, puis est diminuée progressivement au cours des itérations subséquentes selon

diverses stratégies. Pour déterminer la probabilité d'accepter une solution, on utilise généralement la distribution de Boltzmann $e^{\frac{f(s)-f(s')}{T}}$, où $f(s)$ est une fonction d'objectif à minimiser. Lorsque la température est une grande valeur, la probabilité d'acceptation est près de 1. Au fur et à mesure que l'algorithme progresse, la probabilité dépend de plus en plus de la différence de qualité entre la solution courante et la solution voisine. Vers la fin de l'exécution de l'algorithme, la température est tellement basse que seules les solutions de qualité égale ou moindre à la solution courante sont acceptées. En d'autres mots, plus l'algorithme progresse, plus il est apte à identifier de bonnes solutions, mais plus il risque de rester pris dans un optimum local.

La recherche réactive (ou adaptative) contient tout algorithme de recherche locale qui modifie ses paramètres en cours de résolution afin de modifier son comportement pour les itérations subséquentes. Ce changement dans le comportement n'est pas tant un mécanisme pour sortir d'un optimum local, mais plutôt un mécanisme d'apprentissage. La recherche réactive inclut donc les algorithmes tabou et recuit simulé, mais aussi tout algorithme de recherche locale qui apprend de ses "erreurs".

La colonie de fourmis est un bon exemple de méthode adaptative. Dans la réalité, une fourmi quitte sa colonie à la recherche de nourriture. Lorsqu'elle en trouve, elle rebrousse chemin en laissant sur ses pas une substance appelée phéromones que d'autres fourmis peuvent détecter. Les prochaines fourmis qui quittent la colonie peuvent suivre cette trace jusqu'à la source de nourriture sans détours. Plus il y a de nourriture, plus il y aura de fourmis qui passeront par le même chemin, et plus la trace de phéromones sera forte. Les phéromones s'évaporant avec le temps, les chemins les plus courts menant à une source de nourriture seront les plus utilisés. Si un problème peut se formuler comme celui de trouver le plus court chemin menant à un noeud particulier de grande valeur, alors simuler une colonie de fourmis peut résoudre ce problème.

1.4.2.4 Méthodes évolutives

Un algorithme est considéré évolutif si son but est de choisir un ensemble de solutions et de les améliorer en produisant itérativement un nouvel ensemble de solutions de meilleure qualité (si possible) que les ensembles précédents. Chaque ensemble est appelé une “génération”, et chaque génération sert de base pour produire la génération suivante. Dans ce but, on utilise généralement des procédés inspirés de la génétique, comme la reproduction, la mutation, et la sélection naturelle. La reproduction est simulée en sélectionnant deux solutions de la génération précédente et en les combinant d’une quelconque façon afin de produire de nouvelles solutions. La mutation consiste à prendre une solution et à la modifier dans le but de l’améliorer, par exemple, en effectuant quelques itérations d’une recherche locale utilisant la solution comme point de départ. La sélection naturelle se fait simplement en limitant la taille d’une génération, et en ne gardant que les meilleures solutions parmi celles-ci. Ainsi, les solutions les moins bonnes sont éliminées de la “population”, ce qui (selon la théorie de l’évolution) provoque l’amélioration à long terme de l’ensemble de la population.

En général, les algorithmes génétiques utilisent beaucoup l’aléatoire pour générer de nouveaux individus, surtout lors de la reproduction. La recherche dispersée (“scatter search”) introduit des concepts qui diffèrent de la génétique et démontre entre autres qu’il y a des avantages à mieux spécifier comment combiner plusieurs solutions pour en produire de nouvelles, plutôt que d’utiliser des procédés aléatoires. Plusieurs stratégies employées dans la recherche dispersée n’ont aucun lien avec la génétique, ce qui en fait une approche évolutive à part entière.

1.4.2.5 Réseaux de neurones et dynamique du réplicateur

Un réseau de neurones est un concept d’intelligence artificielle qui est utilisé en général pour résoudre des problèmes de reconnaissance de schémas. L’idée est de simuler l’interaction entre une grande quantité de neurones pouvant apprendre à détecter des schémas (“patterns”). Chaque neurone étant une entité indépendante

des autres, ce concept se porte très bien à la résolution en parallèle.

Un algorithme exploitant la dynamique du réplicateur est semblable à la fois à une méthode de population et à un réseau de neurones. On crée une population d'agents indépendants qui possèdent chacun leur stratégie pour atteindre l'objectif du problème. Les stratégies des agents sont itérativement adaptées, en fonction de la concurrence entre les agents, via des équations de réplication (des contraintes forçant les agents à atteindre un équilibre entre eux). On peut faire varier la population en incluant un processus de naissance et de mort des agents. La particularité de ces méthodes est l'absence de règles gérant l'ensemble des agents.

1.4.3 Familles de graphes

Cette catégorie contient tous les algorithmes qui cherchent à déterminer si une instance fait partie d'une famille d'instances connue. Ces familles d'instances ont souvent des propriétés permettant de trouver une solution optimale de manière efficace. On trouve aussi dans cette catégorie des algorithmes qui cherchent le plus grand sous-graphe faisant partie d'une famille de graphes connue. Une méthode heuristique, basée sur cette approche, consiste à résoudre d'abord le problème pour le sous-graphe identifié, produisant ainsi une solution partielle, pour ensuite tenter de trouver une solution au problème original à l'aide de cette solution partielle. Enfin, nous incluons dans cette catégorie tous les algorithmes qui cherchent à simplifier l'instance en la transformant en une autre instance généralement plus facile à résoudre, par exemple, si le graphe modifié fait partie d'une famille de graphes connue.

Pour plus d'informations sur diverses familles de graphes, voir le site Wikipedia : http://en.wikipedia.org/wiki/Category:Graph_families

1.4.4 Programmation non linéaire

Un même problème peut généralement être défini de plusieurs façons. En général, on s'intéresse aux modèles linéaires car il existe une grande quantité d'outils

pour résoudre ces problèmes. Toutefois, il arrive que certains modèles non linéaires (qui ont des contraintes ou un objectif non linéaires) soient tout aussi intéressants à résoudre, par exemple lorsque le domaine réalisable est convexe (toute combinaison convexe de deux solutions réalisables est réalisable). On peut aussi créer des heuristiques basées sur la programmation non linéaire, mais leur fonctionnement diffère quelque peu des heuristiques décrites dans cette thèse.

En général, les méthodes de résolution de problèmes non linéaires ont pour objectif de trouver un optimum local, et s'ils sont chanceux (ou si le problème ne possède qu'un seul optimum), alors cet optimum local est aussi un optimum global. Plusieurs méthodes (gradient, points intérieurs, etc.) partent d'un point réalisable du domaine, trouvent une direction de "descente" (l'objectif s'améliore si l'on progresse dans cette direction), puis déterminent le pas à effectuer dans la direction choisie. Ces informations permettent d'identifier un nouveau point réalisable à partir duquel l'algorithme répète le processus itérativement, jusqu'à ce qu'aucune direction n'améliore l'objectif (la solution courante est donc un optimum).

Pour plus d'informations sur la programmation non linéaire et les méthodes associées, voir [Lue84].

1.4.5 Programmation semi-définie

En programmation semi-définie, l'objectif et les contraintes sont linéaires en fonction des variables, mais ces dernières sont des matrices symétriques semi-définies positives, ce qui fait que la résultante est non linéaire. Toutefois, le domaine réalisable est convexe, ce qui fait qu'il existe des outils permettant de résoudre de tels problèmes, comme la méthode de points intérieurs. La programmation linéaire et la programmation quadratique sont des cas particuliers de la programmation semi-définie [LR05].

1.4.6 Méthodes d'approximation

Cette catégorie contient tous les algorithmes qui, lorsqu'ils se terminent, prouvent que la valeur résultante satisfait un critère d'approximation précis par rapport à la valeur optimale. Par exemple, un algorithme d'approximation pourrait assurer de donner une solution dont la valeur est à au plus $\epsilon > 0$ de la valeur optimale (voir [GJ79] pour une introduction complète à la complexité de divers problèmes et de leur approximation).

1.5 Revue de littérature

Puisque cette thèse discute de trois sujets connexes, mais différents, il en est de même pour la revue de littérature. Les trois prochaines sous-sections décrivent la situation actuelle de la littérature qui a trait de près ou de loin aux sujets traités. Il est toutefois intéressant de mentionner que les problèmes traités dans cette thèse sont tellement difficiles à résoudre que même les approximer est difficile. Il arrive parfois que certains problèmes puissent être approximés plus facilement que résolus à l'optimum. Ce n'est toutefois pas le cas pour les problèmes traités dans cette thèse. Pour une présentation récente à ce sujet, voir [Kho01].

1.5.1 Problème de la clique maximum

Le problème de trouver la plus grande clique d'un graphe a été l'un des premiers problèmes identifiés comme étant NP-difficile. Depuis, un nombre impressionnant d'approches ont été testées pour le résoudre, souvent de façon exacte, mais la plupart du temps de façon heuristique en trouvant une borne inférieure (par exemple, une grande clique) ou bien en trouvant une borne supérieure (par exemple, une coloration). Puisqu'une revue de littérature couvrant l'ensemble du sujet serait hors proportion dans cette thèse, nous référons à la plus récente revue de littérature [BBPP99] pour tout ce qui a trait aux publications qui précèdent 1999. Nous soulignerons toutefois quelques ouvrages majeurs datant d'avant 1999, mais la revue de littérature que nous présentons est basée essentiellement sur les

publications qui ont vu le jour depuis.

L'un des changements majeurs par rapport aux années 1990 est la quasi-disparition des méthodes évolutives. Dans ce contexte, la seule publication à mentionner est celle de Marchiori [Mar02], qui avait auparavant démontré qu'un algorithme génétique simple pouvait être compétitif en comparaison avec des méthodes de recherche locale. Cette publication traite justement des algorithmes génétiques et de recherche locale pour trouver la plus grande clique d'un graphe. De même, l'engouement pour les méthodes basées sur les réseaux de neurones semble avoir diminué. Les contributions de Pelillo et al. [JPR00,PT06], de Bomze et al. [BPS00] ainsi que celle de Wang et al. [WTC03] sont les seules recensées depuis 1999.

Pour ce qui est des méthodes de recherche locale classiques, Watson et al. [WWH05] analysent les mécanismes tabous afin d'en extraire une explication concernant l'efficacité (ou l'absence d'efficacité) sur certaines instances, en particulier pour le problème de "job-shop" (un problème d'affectation de tâches à des machines). Une approche nouvelle de Katamaya et al. [KHN04], appelée $k - opt$, consiste à effectuer des recherches dans des voisinages de taille variable afin d'agrandir l'espace de recherche et ainsi permettre de trouver une solution à l'extérieur d'un optimum local. Enfin, Bomze et al. [BBPR02] se sont inspirés du recuit simulé et des travaux récents sur la dynamique du réplicateur.

Un nouveau type de recherche locale a vu le jour, basée sur l'historique des mouvements précédents, dont les ancêtres sont certainement les approches de recherche locale tabou [Glo89] et colonie de fourmis [DC99]. D'abord appelée recherche réactive, elle s'est ensuite développée pour englober toute méthode qui adapte son comportement, ce qui lui confère le nom de méthode adaptative. Parmi les premières contributions, l'on trouve celles de Jagota et Sanchis [JS01], de Battiti et Protasi [BP01], ainsi que le mémoire de maîtrise de l'auteur de cette thèse [SL02], basé sur un travail de St-Louis, Ferland et Gendron [SLFG06]. Les travaux les plus récents [GLC04,SF06,PH06] démontrent bien la puissance de cette approche.

Un aspect peu étudié du problème est la résolution en temps réel, ce qui demande une capacité de trouver la taille de la plus grande clique très rapide-

ment (ou du moins une valeur s'en approchant le plus possible). On utilise en général un algorithme glouton tel que MIN [HRS02], qui construit une clique en sélectionnant itérativement le noeud de plus grand degré, mais d'autres stratégies peuvent s'avérer utiles [Kum06].

Puisque plusieurs décennies se sont écoulées depuis l'introduction de la problématique, de nouvelles façons de modéliser le problème peuvent s'avérer utiles en ouvrant des voies de recherche inexplorées. Le modèle mathématique introduit précédemment utilise des variables binaires, ce qui implique que l'espace des solutions est contenu dans un hypercube unitaire. Busygin, Butenko et Pardalos [BBP02] ont développé une heuristique qui cherche une solution dans un espace sphérique (au lieu de l'hypercube). Récemment, Budinich et Budinich [BB06] ont introduit un modèle mathématique dans un autre contexte, appelé "spinorial", qui utilise principalement la notion de rotation géométrique sur les vecteurs, afin de profiter des progrès récents dans ce domaine.

Pour résoudre exactement le problème de la plus grande clique, on utilise généralement l'algorithme de branch-and-bound, dont l'une des implémentations les plus efficaces est certainement celle de Carraghan et Pardalos [CP90], amélioré récemment par Östergård [Öst02]. Pour d'autres contributions basées sur le branch-and-bound, voir [Woo97, Sew98, Kum04, BT90]. En général, la taille du graphe a une grande influence sur le temps requis par les méthodes exactes pour obtenir une solution optimale. Toute transformation [Her00, GH01] permettant de réduire cette taille s'avère donc intéressante.

Une autre approche pour résoudre exactement le problème est la programmation par contraintes. Les premiers travaux dans cette branche ont été introduits par Fahle [Fah02], qui ordonne plusieurs bornes supérieures selon leur efficacité afin d'utiliser celle qui donnera les meilleurs résultats. De même, Régim [Rég03] obtient d'excellents résultats et introduit une nouvelle borne supérieure basée sur un problème d'affectation.

L'étude des bornes supérieures et inférieures a un impact direct sur l'efficacité des méthodes de branch-and-bound et de programmation par contraintes. Toutes

les méthodes heuristiques mentionnées ci-haut tentent de trouver une bonne borne inférieure (une grande clique) au problème. Une bonne borne supérieure est beaucoup plus complexe à obtenir. La meilleure connue à ce jour est certainement obtenue en calculant un invariant du graphe appelé le nombre de Lovász [Lov79, Knu94], qui est une valeur comprise entre la taille de la plus grande clique du graphe et le nombre chromatique, et qui est calculable en un temps polynomial en fonction du nombre de noeuds dans le graphe. Cette valeur est toutefois complexe à calculer, car elle est basée sur la programmation semi-définie, et la méthode la plus efficace connue pour résoudre ce problème est la méthode de points intérieurs. En pratique, résoudre ce problème est très lent, et se prête mal à une intégration dans un algorithme de branch-and-bound. On utilisera plutôt un algorithme rapide de coloration de graphes pour obtenir les bornes en question. Pour d'autres contributions concernant des bornes sur la plus grande clique d'un graphe, voir [Bud03, Aou06].

Pour comparer des méthodes entre elles, il est utile d'avoir un ensemble d'instances variées, comprenant des petits et grands graphes, de densités différentes et de structures différentes. En 1992, le concours DIMACS [JT93] a été l'instigateur d'une telle banque d'instances. Plusieurs y ont apporté leur contribution, dont Brockington et Culberson [BC96], d'où proviennent les célèbres "brock" qui font partie des instances les plus difficiles du DIMACS. Depuis 1999, toutefois, il existe peu de contributions dans ce domaine, sauf celle de Busygin [Bus06], qui étudie une famille de graphes donnant du fil à retordre à un algorithme glouton simple.

Trouver la plus grande clique pondérée [BCN87], où chaque noeud possède un poids et la solution optimale est la clique de plus grand poids, est une généralisation du problème de la plus grande clique. Plusieurs approches exactes [Bab94] et heuristiques [BN98] ont été développées pour résoudre ces problèmes, ayant en général de grandes similarités avec les méthodes développées pour le problème standard.

1.5.2 Problème de la coloration minimum

Le problème de trouver le nombre chromatique d'un graphe fait partie des premiers problèmes ayant été démontrés NP-difficiles. Un grand nombre de problèmes

pratiques peuvent être résolus par un algorithme de coloration de graphes, ce qui fait que la littérature à ce sujet est abondante. Seule une sélection des ouvrages les plus appropriés est présentée ici. Il faut noter que plusieurs revues de la littérature sur la coloration de graphes et les problèmes reliés ont été publiées dans le passé. Nous référons aux articles [JT95, PMX98, GH06, Chi06] ainsi qu'à la page Web de Culberson [Cul06] pour un complément à la revue présente.

Plusieurs études portent sur la complexité du problème [Law76, Mit02] en fonction de la structure du graphe [Epp01], ou bien sa k -colorabilité [AF99, BCMM06] ou sa non k -colorabilité [EDM03], en particulier sur des graphes aléatoires [AK97]. Les propriétés des graphes aléatoires sont particulièrement étonnantes [Kri02b, Kri02a, MPWZ02] puisque le nombre chromatique de ces graphes est en général prévisible avec une grande précision. Cette propriété justifie donc que l'on tente de générer des graphes autrement qu'aléatoirement [BFP01], pour éviter cette prévisibilité. L'étude des graphes critiques [Kri97], dont le nombre chromatique est inférieur pour tout sous-graphe induit, est particulièrement utile en pratique [HH02b, DGH04]. Une autre étude intéressante et récente est celle où l'on examine pourquoi certains noeuds du graphe partagent toujours la même couleur dans toute solution optimale [CG01, HH06a]. En effet, la détection efficace de ces noeuds rendrait la résolution du problème beaucoup plus aisée.

Les plus anciennes méthodes exactes sont basées sur une énumération implicite des solutions réalisables [Bro72, KJ85]. Mais ce n'est plus le cas depuis l'introduction des algorithmes de branch-and-bound [Bré79, Pee83, Sew96], de branch-and-cut [MDZ06], de branch-and-price [HLS05] et de génération de colonnes [MT96]. De récentes contributions [HH02b, DGH04] démontrent qu'il est possible de combiner les forces d'une méthode exacte et d'une heuristique pour obtenir une nouvelle méthode exacte efficace en utilisant l'heuristique pour déterminer un sous-graphe sur lequel la méthode exacte est utilisée. En réduisant ainsi la taille du graphe, le temps requis pour trouver une solution optimale en est réduit. Toutefois, il reste que toutes ces méthodes ne peuvent être utilisées que sur des instances relativement petites. Pour toutes les autres instances, il ne reste que des heuristiques pour

parvenir à les résoudre en des temps raisonnables.

L'une des premières heuristiques efficaces pour colorer un graphe, introduite en 1979, est DSATUR [Bré79], un algorithme glouton qui sert encore de nos jours à obtenir rapidement une coloration. Il y a peu d'algorithmes plus simples que DSATUR, outre la coloration séquentielle, qui consiste à colorer chaque noeud en utilisant la plus petite couleur possible et ce, dans l'ordre lexicographique. Il est tout de même intéressant de noter qu'il existe toujours une façon de changer l'ordre lexicographique de façon à ce que la coloration séquentielle donne une coloration optimale. Le problème de la coloration de graphes se réduit donc à trouver le "bon" ordre lexicographique ; c'est pourquoi on associe souvent le problème de coloration de graphes à un problème d'ordonnancement de tâches. Pour d'autres approches déterministes, voir [Veg99, Klo02].

La même année, une publication étonnante de Lovász [Lov79] introduit la fonction θ , permettant de calculer un invariant situé entre le nombre chromatique et la plus grande clique du graphe (voir sous-section précédente). Cette valeur borne donc deux problèmes NP-difficiles, tout en étant calculable en temps polynomial. Pour des travaux subséquents à propos de cette fonction, voir [Knu94, BY99, CO06]. D'autres travaux [KV00] ont été réalisés afin d'approximer le nombre chromatique et la taille du plus grand stable d'un graphe, un problème équivalent à celui de trouver la plus grande clique dans le complément du graphe.

Quelques années ont passé avant l'arrivée des premières heuristiques de recherche locale, qui sont d'ailleurs toujours utilisées de nos jours ; par exemple, la méthode de recuit simulé [CHdW87, BJH03] et la recherche avec tabous [HdW87, Glo89, GVL02]. La coloration par une colonie de fourmis [CH97, VZ00], est apparue une dizaine d'années plus tard. De ces approches sont nées les heuristiques réactives [Blo01, BZ03] et adaptatives [GHZ03].

Pour certaines instances, il arrive que l'on épuise toutes les approches disponibles sans obtenir de solution satisfaisante. On doit alors se tourner vers des combinaisons de méthodes [FH00, Wal01, AKL02, MMT05, CHD95, Her03], qui donnent souvent d'excellents résultats, combinant les forces de chaque approche tout en

évitant autant que possible leurs mauvais côtés. On peut penser à faire varier la définition de voisinage [AHZ03] lorsqu'on semble stagner, par exemple en utilisant un voisinage de taille exponentielle [GPB05] (malgré des résultats préliminaires décevants, selon les auteurs). La recherche locale itérative [RLMS00, PS02a, CS02, CDS03] analyse la solution obtenue par différents algorithmes de recherche locale afin de s'orienter pour les itérations subséquentes. Cette approche offre une très grande flexibilité selon l'impact que l'on donne à chaque solution et au niveau de modification permis de la solution courante. Une étude des méthodes exactes de type énumérative comparées aux méthodes de recherche locale a permis d'obtenir une hybridation tout aussi intéressante [GPR96, ZZ96, Pre02a, Pre01].

On peut aussi incorporer des méthodes de recherche locale à l'intérieur d'algorithmes évolutifs [GHZ03]. L'un des meilleurs exemples est certainement celui de Galinier et Hao [GH99], qui est encore compétitif de nos jours. Pour d'autres exemples, voir [FF96, FL97, DH98, MPBG99, MD00, FLS01, GPB03]. Aussi, plusieurs méthodes se sont inspirées d'autres aspects de la génétique, entre autres [Gol06, CNP03]. Quelques exemples de recherche dispersée donnent de bons résultats [HH02a, HH06b, GLM03].

Une autre approche, basée sur la physique cette fois, est la programmation quantique [BBBV97, HP00, FH02]. La notion d'état quantique est utilisée pour décrire la mécanique à l'intérieur des atomes qui constituent toute la matière de l'univers. Un ordinateur quantique est une machine qui est capable de contrôler les états quantiques de la matière pour des fins de calcul. Sa puissance provient du fait que l'on peut évaluer simultanément toutes les permutations possibles d'un grand nombre d'éléments. Un tel outil pourrait être utilisé pour résoudre des problèmes en combinatoire et en théorie des graphes, si ce n'était du fait que cette machine n'existe qu'à l'état théorique pour l'instant, car de nombreux obstacles empêchent encore sa construction dans la vie réelle.

L'une des approches les plus récentes est la coloration par orientation d'arêtes. L'idée fondamentale est basée sur un théorème énoncé indépendamment par Galai [Gal68], Roy [Roy67], et Vitaver [Vit62], qui assure que le nombre chromatique

est égal au minimum, parmi toutes les orientations sans circuits du graphe, du nombre de noeuds d'un chemin de longueur maximale. Quelques ouvrages [Sta73, Dem79, Vo87, BS99, Las01] étudiant les propriétés des orientations d'un graphe ont pavé la voie jusqu'au tournant du siècle, où les premières heuristiques commencent à voir le jour, par exemple Barbosa et al. [BF06]. Une approche similaire était étudiée depuis certains temps pour le problème d'ordonnancement de tâches [Bar00], qui est un problème somme toute assez semblable à la coloration de graphes. Le pas semble évident entre les deux problèmes, mais personne ne semblait avoir l'idée de s'y aventurer. Il est intéressant de noter que trouver l'orientation qui maximise (au lieu de minimiser) la longueur du plus long chemin correspond au problème de trouver le plus long chemin dans un graphe non orienté. Ce problème est aussi NP-difficile car il permet de résoudre le problème NP-complet du chemin Hamiltonien, qui consiste à déterminer s'il existe un chemin parcourant tous les noeuds du graphe une seule fois.

1.5.3 Problème de la coloration par bande et autres généralisations

Colorer un graphe ne signifie pas toujours que l'on veut donner une couleur à chaque noeud tout en s'assurant que chaque arête relie deux noeuds de couleur différente. En effet, il existe de multiples variantes et généralisations, et les méthodes pour résoudre ces problèmes varient tout autant. Pour une énumération des variantes les plus courantes, nous référons au site Wikipedia :

http://en.wikipedia.org/wiki/Graph_coloring.

On considère qu'un problème X est une généralisation de la coloration des noeuds d'un graphe si cette dernière est un cas particulier du problème X. Un exemple de généralisation est la coloration par bande ("bandwidth coloring"), qui consiste à colorer les noeuds d'un graphe ayant des poids non-négatifs sur chaque arête, de sorte que la différence entre la couleur de deux noeuds adjacents doit être supérieure ou égale au poids de l'arête les reliant, et où l'objectif est de minimiser la plus grande couleur utilisée. La coloration "standard" des noeuds d'un graphe correspond au cas particulier où toutes les arêtes ont un poids égal à 1. Cette

généralisation provient du problème d'affectation de signal radio, où le poids sur chaque arête représente la distance minimale, en termes de fréquence, entre des signaux en conflit afin d'éviter de l'interférence. La multicoloration ("multicoloring") est une autre généralisation consistant à colorer les noeuds d'un graphe mais où chaque noeud doit être coloré en utilisant un nombre pré-défini de couleurs tout en s'assurant que pour chaque paire de noeuds reliés par une arête, l'intersection des ensembles de couleurs est vide. L'objectif de la multicoloration est de minimiser le nombre de couleurs utilisées. La coloration "standard" des noeuds d'un graphe correspond au cas particulier où tous les noeuds doivent être colorés en n'utilisant qu'une seule couleur. Enfin, la multicoloration par bande, comme son nom l'indique, combine la multicoloration et la coloration par bande, c'est-à-dire que chaque noeud doit être coloré en utilisant un nombre pré-défini de couleurs, et où pour chaque arête du graphe, la différence entre toute paire de couleurs des noeuds reliés par l'arête doit être supérieure ou égale au poids de l'arête.

En 2002, une conférence nommée "COLOR02" a réuni des spécialistes internationaux afin de discuter des résultats les plus récents au sujet de la coloration de graphe et de ses généralisations, une sorte de suite à celle du DIMACS en 1992 [JT93], et qui fut répétée pendant trois années consécutives. Le site Web associé à l'événement (<http://mat.gsia.cmu.edu/COLORING03/>) contient une grande quantité d'informations pertinentes pour toutes les généralisations mentionnées ci-haut, incluant des instances de graphes sur lesquelles il est possible de tester de nouvelles approches pour colorer un graphe. En particulier, nous nous intéressons aux instances de type "GEOM", fournies par M. Trick, et qui contiennent des graphes de 20 à 120 noeuds pour le problème de coloration par bande.

Lim et al. [LZZ03] ont produit un ouvrage décrivant ces généralisations, introduisant une hybridation d'une méthode de SWO ("Squeaky Wheel Optimization") [JC98] et d'un algorithme glouton itératif (IG) [CL96] pour résoudre toutes ces généralisations de manière compétitive. Plusieurs de ces généralisations proviennent du domaine de la téléphonie radio, où la structure théorique derrière les

conflits et interférences est très proche de la coloration de graphes (voir [AvHK⁺03, HSZ05] pour des exemples). Pour le problème de coloration par bande, Phan and Skiena [PS02b] ont utilisé une implémentation d'une heuristique de descente afin d'obtenir une borne intéressante sur les instances "GEOM". Pour plus d'exemples concernant la multicoloration, voir [HK02, NS01, Nar02]. Enfin, pour résoudre la multicoloration par bande, Prestwich [Pre02b] a combiné une méthode exacte énumérative ainsi qu'une recherche locale nommée IMPASSE, et a démontré la robustesse de l'approche en particulier sur les instances "GEOM".

CHAPITRE 2

MÉTHODE DE DÉCOMPOSITION POUR LE PROBLÈME DE LA CLIQUE MAXIMUM

Les bornes supérieures sur la plus grande clique d'un graphe sont difficiles à obtenir. On utilise généralement une heuristique de coloration, puisque le résultat est une borne supérieure sur le nombre chromatique, qui est à son tour une borne supérieure sur la taille de la plus grande clique. La meilleure borne supérieure connue est certainement le nombre de Lovasz, une valeur comprise entre la taille de la plus grande clique et le nombre chromatique. Toutefois, cette valeur est si complexe à évaluer qu'elle n'est généralement pas utilisée en pratique. Nous présentons ici une méthode de décomposition qui sert à améliorer n'importe quelle méthode de borne supérieure en l'utilisant comme sous-routine pour calculer des bornes sur des sous-graphes. Les résultats donnent une amélioration moyenne de 60% lorsqu'on incorpore une méthode de coloration de graphes, et donne même une amélioration de 100% pour quelques instances (on prouve alors l'optimalité pour ces instances). De plus, en observant les dernières itérations de notre méthode de décomposition, on peut extraire un sous-graphe qui contient en général la plus grande clique du graphe. En tentant de trouver la plus grande clique dans ce sous-graphe, on obtient alors une borne inférieure sur la plus grande clique du graphe en entier.

Les deux prochaines sections expliquent notre approche, suivi de l'article que nous soumettrons sous peu pour publication et qui contient tous les détails des preuves et les résultats obtenus sur des instances largement utilisées dans le domaine.

2.1 Méthode de décomposition

Étant donné une méthode produisant une borne supérieure sur la taille de la plus grande clique d'un graphe G , dénotée $h(G)$, une décomposition simple consisterait

à calculer, pour tout noeud v du graphe, la valeur de la fonction $h(N_G(v))$, où $N_G(v)$ est le sous-graphe induit par le noeud v et tous les noeuds qui y sont reliés par une arête. La plus grande valeur ainsi calculée serait une borne supérieure sur la plus grande clique de G , puisque toute clique C de G fait partie du sous-graphe $N_G(v)$ pour tout noeud $v \in C$. On peut aussi appliquer le même processus en calculant la borne sur tous les sous-graphes induits par les arêtes du graphe (au lieu des noeuds). En fait, on peut généraliser à toute clique de taille $k \geq 1$ du graphe, mais le temps de calcul croît alors de manière exponentielle car l'algorithme doit énumérer itérativement toutes les cliques de taille k du graphe, ce qui rend la généralisation peu attrayante en pratique.

Notre méthode de décomposition est fortement inspirée de la décomposition simple par noeuds, décrite au paragraphe précédent. Tout comme cette dernière, on calcule pour chaque noeud du graphe la valeur de la borne supérieure sur le sous-graphe induit par ce noeud et son voisinage (les noeuds qui y sont reliés). Nous introduisons ensuite une variable appelée "borne candidate", initialisée à la plus petite valeur calculée sur tous les noeuds du graphe, et qui contiendra éventuellement notre borne supérieure finale. Une fois notre borne candidate initialisée, nous posons la question suivante : existe-t-il une clique de taille supérieure à cette valeur ? Si la réponse est non, alors la valeur de notre borne candidate est effectivement une borne supérieure sur la taille de la plus grande clique dans le graphe. Par contre, si la réponse est oui, alors tous les noeuds du graphe dont la valeur de la borne est inférieure ou égale à notre borne candidate ne peuvent faire partie d'une clique de taille plus grande que celle-ci. On peut donc les supprimer du graphe avant de poursuivre la recherche. Or, une fois ces noeuds supprimés, il se peut que les bornes supérieures associées à d'autres noeuds du graphe deviennent à leur tour inférieures ou égales à notre borne candidate ; ces noeuds pourront être à leur tour supprimés.

Le processus de suppression se poursuit tant qu'il existe des noeuds dont la valeur de la borne (mise à jour) est inférieure ou égale à notre borne candidate, et qu'il existe au moins un noeud dont la valeur de la borne (mise à jour) est strictement supérieure à notre borne candidate. Lorsque le processus s'arrête, alors

soit que le graphe a été entièrement supprimé ou que tous les noeuds ont une valeur inférieure ou égale à notre borne candidate (auquel cas la réponse à la question était non, et notre borne candidate est effectivement une borne supérieure sur la taille de la plus grande clique du graphe), ou bien tous les noeuds ont une borne strictement supérieure à notre borne candidate. Dans ce cas, nous la mettons à jour en utilisant la nouvelle plus petite valeur des bornes supérieures associées aux noeuds restants et nous recommençons le processus de suppression.

L'algorithme de décomposition alterne donc entre le processus de suppression et la mise à jour de la borne candidate, jusqu'à ce que tous les noeuds du graphe soient supprimés ou que leur valeur soit inférieure ou égale à notre borne candidate. Cette dernière est alors une borne supérieure sur la taille de la plus grande clique. Pour s'en convaincre, il suffit de constater qu'un noeud n'est supprimé que lorsque sa borne est inférieure ou égale à la borne candidate (qui est strictement croissante). En particulier, un noeud de la plus grande clique C du graphe n'est supprimé que lorsque $|C|$ est inférieur ou égal à la valeur de la borne candidate. Alors si notre borne candidate a une valeur strictement inférieure à $|C|$, c'est qu'aucun noeud de la clique n'a été supprimé. Mais dans ce cas, la valeur de la borne pour chaque noeud de la plus grande clique est strictement supérieure à notre borne candidate, ce qui contredit le critère d'arrêt. On en conclut donc que la valeur finale de notre borne candidate est forcément supérieure ou égale à $|C|$.

La complexité de notre méthode de décomposition est dans l'ordre de $|V|^2$, multiplié par la complexité du calcul de $h(G)$. C'est la même complexité que la décomposition simple sur les arêtes, mais en pratique, notre méthode est beaucoup plus rapide que celle-ci puisque le graphe diminue strictement à chaque itération. De plus, les résultats démontrent que notre méthode produit en général une borne plus stricte, ce qui en fait une approche beaucoup plus intéressante à utiliser que la décomposition par arête. Pour ce qui est de la décomposition simple par noeud, sa complexité est inférieure d'un ordre de grandeur, mais la borne ainsi obtenue est toujours supérieure ou égale à la borne obtenue en utilisant notre décomposition. Quand la qualité de la borne est un facteur important, notre décomposition devient

donc d'autant plus intéressante.

2.2 Extraction d'un sous-graphe pour le calcul d'une borne inférieure

Puisque les premiers noeuds supprimés par notre décomposition sont ceux qui ont la plus petite borne, il est tout naturel de supposer que la chance que ces noeuds fassent partie d'une clique de taille maximale est moindre que pour les noeuds qui sont supprimés en dernier. Si l'on considère le graphe résiduel au moment de la dernière mise à jour de notre borne candidate (autrement dit, lorsqu'elle atteint sa valeur définitive), on obtient un sous-graphe ayant une grande chance de contenir une clique de taille maximum du graphe original. C'est en utilisant ce sous-graphe que nous calculerons une borne inférieure sur la taille de la plus grande clique.

En effet, calculer la taille de la plus grande clique dans un sous-graphe nous donne une borne inférieure sur la taille de la plus grande clique du graphe en entier. L'avantage de notre méthode de décomposition est qu'elle permet d'identifier un sous-graphe ayant de fortes chances de contenir la plus grande clique du graphe. Les résultats démontrent que la plupart du temps, la plus grande clique du graphe se trouve effectivement dans ce sous-graphe.

**A DECOMPOSITION ALGORITHM FOR COMPUTING
BOUNDS ON THE CLIQUE NUMBER OF A GRAPH**

Bernard Gendron

Département d'informatique
et de recherche opérationnelle
and

Centre de recherche sur les transports
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal, Québec H3C 3J7

Alain Hertz

Département de mathématiques et de génie industriel
École Polytechnique de Montréal
C.P. 6079, succ. Centre-ville
Montréal, Québec H3C 3A7
and

GERAD

3000, chemin de la Côte-Sainte-Catherine
Montréal, Québec H3T 2A7

Patrick St-Louis

Département d'informatique
et de recherche opérationnelle
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal, Québec H3C 3J7

Abstract

We consider the problem of determining the size of a maximum clique in a graph, also known as the clique number. Given any method that computes an upper bound on the clique number of a graph, we present a decomposition algorithm which is guaranteed to improve upon that upper bound. Computational experiments on DIMACS instances show that, on average, this algorithm can reduce the gap between the upper bound and the clique number by about 60%. We also show how to use this decomposition algorithm to improve the computation of lower bounds on the clique number of a graph.

2.3 Introduction

In this paper, we consider only undirected graphs with no loops or multiple edges. For a graph G , we denote $V(G)$ its vertex set and $E(G)$ its edge set. The *size* of a graph is its number of vertices. The subgraph of G induced by a subset $V' \subseteq V(G)$ of vertices is the graph with vertex set V' and edge set $\{(u, v) \in E(G) \mid u, v \in V'\}$. A *complete graph* is a graph G such that u and v are adjacent, for each pair $u, v \in V(G)$. A *clique* of G is an induced subgraph that is complete. The *clique number* of a graph G , denoted $\omega(G)$, is the maximum size of a clique of G . Finding $\omega(G)$ is known as the *clique number problem*, while finding a clique of maximum size is the *maximum clique problem*. Both problems are NP-hard [GJ79]. Many algorithms, both heuristic and exact, have been designed to solve the clique number and maximum clique problems, but finding an optimal solution in relatively short computing times is realistic only for small instances. The reader may refer to [BBPP99] for a survey on algorithms and bounds for these two problems.

An upper bound on the clique number of a graph is useful in both exact and heuristic algorithms for solving the maximum clique problem. Typically, upper bounds are used to guide the search, prune the search space and prove optimality. One of the most famous upper bounds on the clique number of a graph G is the *chromatic number*, $\chi(G)$, which is the smallest integer k such that a legal k -coloring exists (a legal k -coloring is a function $c : V(G) \rightarrow \{1, 2, \dots, k\}$ such that $c(u) \neq c(v)$ for all edges $(u, v) \in E(G)$). Finding the chromatic number is known as the *graph coloring problem*, which is NP-hard [GJ79]. Since $\chi(G) \geq \omega(G)$, any heuristic method for solving the graph coloring problem provides an upper bound on the clique number. Other upper bounds on the clique number will be briefly discussed in Section 2.6 (for an exhaustive comparison between the clique number and other graph invariants, see [Aou06]).

In this paper, we introduce a decomposition algorithm which makes use of the *closed neighborhood* $N_G(u)$ of any vertex $u \in V(G)$, defined as the subgraph of G

induced by $\{u\} \cup \{v \in V(G) \mid (u, v) \in E(G)\}$. Given an arbitrary upper bound $h(G)$ on $\omega(G)$, the proposed algorithm produces an upper bound $h^*(G)$ based on the computation of $h(N_{G'}(u))$ for a series of subgraphs G' of G and vertices $u \in V(G')$. Under mild assumptions we prove that $\omega(G) \leq h^*(G) \leq h(G)$. As reported in Section 2.6, our tests on DIMACS instances [JT93] show that embedding a graph coloring heuristic (i.e., $h(G)$ is an upper bound on $\chi(G)$ produced by a heuristic) within this decomposition algorithm reduces the gap between the upper bound and $\omega(G)$ by about 60%.

In Section 2.4, we present the decomposition algorithm in more details and we prove that it provides an upper bound on the clique number of a graph. In Section 2.5, we discuss how the decomposition algorithm can also be used to improve the computation of lower bounds on the clique number. We present computational results on DIMACS instances in Section 2.6, along with concluding remarks.

2.4 The Decomposition Algorithm

Assuming $h(G')$ is a function that provides an upper bound on the clique number of any induced subgraph G' of G (including G itself), the decomposition algorithm is based on computing this upper bound for a series of subgraphs G' of G . By computing this upper bound for the closed neighborhood $N_G(u)$ of each vertex $u \in V(G)$, one can easily get a new upper bound $h^1(G)$ on $\omega(G)$, as shown by the following proposition.

Proposition 1. $\omega(G) \leq \max_{u \in V(G)} h(N_G(u)) \equiv h^1(G)$.

Proof. Let v be a vertex in a clique of maximum size of G . This implies that $N_G(v)$ contains a clique of size $\omega(G)$. Hence, $\omega(G) = \omega(N_G(v)) \leq h(N_G(v)) \leq \max_{u \in V(G)} h(N_G(u))$. \square

Now let s be any vertex in $V(G)$, and let G_s denote the subgraph of G induced by $V(G) \setminus \{s\}$. We then have $\omega(G) = \max\{\omega(N_G(s)), \max_{u \in V(G_s)} \omega(N_{G_s}(u))\}$. This equality is often used in branch and bound algorithms for the computation of the

clique number of G (see for example [PX94]). By using function h to compute an upper bound on the clique number of $N_G(s)$ as well as on the clique number of the closed neighborhoods of the vertices of G_s , we can obtain another upper bound $h_s^2(G)$ on $\omega(G)$, as demonstrated by the following proposition.

Proposition 2. $\omega(G) \leq \max\{ h(N_G(s)), \max_{u \in V(G_s)} h(N_{G_s}(u)) \} \equiv h_s^2(G) \quad \forall s \in V(G)$.

Proof. Consider any vertex $s \in V(G)$. If s belongs to a clique of size $\omega(G)$ in G , then $\omega(G) = \omega(N_G(s)) \leq h(N_G(s)) \leq h_s^2(G)$. Otherwise, there is a clique of size $\omega(G)$ in G_s . By Proposition 1 applied to G_s , we have $\omega(G) = \omega(G_s) \leq \max_{u \in V(G_s)} h(N_{G_s}(u)) \leq h_s^2(G)$. \square

Given the graph G_s , one can repeat the previous process and select another vertex to remove, proceeding in an iterative fashion. This gives the *decomposition algorithm* of Figure 1, which provides an upper bound $h^*(G)$ on the clique number of G .

Decomposition algorithm

1. Set $G' \leftarrow G$, $G^* \leftarrow G$ and $h^*(G) \leftarrow 0$;
2. Select a vertex $s \in V(G')$;
 If $h^*(G) < h(N_{G'}(s))$ then set $h^*(G) \leftarrow h(N_{G'}(s))$ and $G^* \leftarrow G'$;
3. If $h^*(G) < \max_{u \in V(G')} h(N_{G'}(u))$ then set $G' \leftarrow G'_s$ and go to 2;
 Else STOP : return $h^*(G)$ and G^* .

Figure 1. The decomposition algorithm

Notice that the decomposition algorithm also returns the subgraph G^* of G for which $h^*(G)$ was updated last. This subgraph will be useful for the computation of a lower bound on $\omega(G)$, as shown in the next section.

Proposition 3. *The decomposition algorithm is finite and its output $h^*(G)$ is an upper bound on $\omega(G)$.*

Proof. The algorithm is finite since at most $|V(G)| - 1$ vertices can be removed from G before the algorithm stops. Indeed, if the algorithm enters Step 2 with a unique vertex s in $V(G')$, then $h^*(G) \geq h(N_{G'}(s)) = \max_{u \in V(G')} h(N_{G'}(u))$ at the end of this Step, and the stopping criterion of Step 3 is satisfied.

Let $W \subseteq V(G)$ denote the set of vertices that belong to a maximum clique in G and let G' denote the remaining subgraph of G when the algorithm stops. If $W \subseteq V(G')$, then we know from Proposition 1 applied to G' that $\omega(G) = \omega(G') \leq \max_{u \in V(G')} h(N_{G'}(u)) \leq h^*(G)$. So assume $W \cap (V(G) \setminus V(G')) \neq \emptyset$. Let s be the first vertex in W that was removed from G , and let G'' denote the subgraph of G from which s was removed. Just after removing s , we have $\omega(G) = \omega(G'') = \omega(N_{G''}(s)) \leq h(N_{G''}(s)) \leq h^*(G)$. Clearly, $h^*(G)$ cannot decrease in the remaining iterations, which yields the conclusion. \square

Under the mild assumption that the upper bound function h is *increasing*, i.e., $h(G') \leq h(G)$ whenever $G' \subseteq G$, we can order the bounds determined by the last three propositions and compare them to $h(G)$, the upper bound computed on G itself.

Proposition 4. *Let s be the vertex selected at the first iteration of the decomposition algorithm. If h is an increasing function, then we have $\omega(G) \leq h^*(G) \leq h_s^2(G) \leq h^1(G) \leq h(G)$.*

Proof. The first inequality, $\omega(G) \leq h^*(G)$, was proved in Proposition 3. To prove the second inequality, $h^*(G) \leq h_s^2(G)$, consider the iteration of the decomposition algorithm where $h^*(G)$ was updated last. If this last update happened at the first iteration, we have $h^*(G) = h(N_G(s)) \leq h_s^2(G)$. Otherwise, let s' be the vertex selected for the last update of $h^*(G)$, and let G^* denote the subgraph in which s' was selected. We have $N_{G^*}(s') \subseteq N_{G_s}(s')$ which gives

$$h^*(G) = h(N_{G^*}(s')) \leq h(N_{G_s}(s')) \leq \max_{u \in V(G_s)} h(N_{G_s}(u)) \leq h_s^2(G).$$

The inequality $h_s^2(G) \leq h^1(G)$ follows from the fact that G_s is a subgraph of G . Indeed, $h_s^2(G) = \max\{h(N_G(s)), \max_{u \in V(G_s)} h(N_{G_s}(u))\} \leq \max\{h(N_G(s)), \max_{u \in V(G_s)} h(N_G(u))\} = \max_{u \in V(G)} h(N_G(u)) = h^1(G)$. Finally, the inequality $h^1(G) \leq h(G)$ is a direct consequence of the hypothesis that h is increasing, since the closed neighborhood of any vertex of G is an induced subgraph of G . \square

If h is not increasing, the relationships above between the different bounds do not necessarily hold. Consider, for instance, the following upper bound function:

$$h(G) = \begin{cases} \chi(G) & \text{if } |V(G)| \geq 4 \\ |V(G)| & \text{otherwise.} \end{cases}$$

Using this function on a square graph (4 vertices, 4 edges, organised in a square) gives $h(G) = \chi(G) = 2$, while the upper bound computed on the closed neighborhood of any vertex u gives $h(N_G(u)) = |V(N_G(u))| = 3 > h(G)$, which implies $h^*(G) = h_s^2(G) = h^1(G) = 3 > 2 = h(G)$.

Nonetheless, even when h is not increasing, it is easy to modify the bound definitions to obtain the result of the last proposition. For instance, one can replace each bound $h(N_{G'}(u))$ by $\min\{h(G), h(N_{G'}(u))\}$. In practice, this implies that we add computing time at the start of the decomposition algorithm to determine $h(G)$, only to prevent an event that is unlikely to happen. This is why we chose not to incorporate this safeguard into our implementation of the decomposition algorithm.

To fully describe the decomposition algorithm, it remains to specify how to select the vertex s to be removed at every iteration. Since the decomposition algorithm updates the value of $h^*(G)$ by setting $h^*(G) \leftarrow \max\{h^*(G), h(N_{G'}(s))\}$, we select the vertex s that minimizes $h(N_{G'}(u))$ over all $u \in V(G')$.

Figure 2 illustrates all bounds when using $h(G) = |V(G)|$. We indicate the value $h(N_{G'}(u))$ for every graph G' and for every vertex $u \in V(G')$. We obviously have $h(G) = 7$. As shown in Figure 2a, $h(N_G(a)) = 5$, $h(N_G(b)) = h(N_G(c)) = 4$,

and $h(N_G(u)) = 2$ for $u = d, e, f, g$, and we therefore have $h^1(G) = 5$. Also, we see from Figure 2b that $h(N_{G_b}(a)) = 4$, $h(N_{G_b}(c)) = 3$, $h(N_{G_b}(u)) = 2$ for $u = d, e, f$, and $h(N_{G_b}(g)) = 1$, and this gives an upper bound $h_b^2(G) = 4$. The decomposition algorithm is illustrated in Figure 2c. The black vertices correspond to the selected vertices. At the first iteration, one can choose $s = d, e, f$ or g , say d and this gives value 2 to $h^*(G)$. Then vertices e, f and g are removed without modifying $h^*(G)$. Finally, the algorithm selects one of the three vertices in the remaining graph G' , say a , and stops since $h^*(G)$ is set equal to $3 = h(N_{G'}(a)) = h(N_{G'}(b)) = h(N_{G'}(c))$. The final upper bound is therefore $h^*(G) = 3$, which corresponds to the size of the maximum clique.

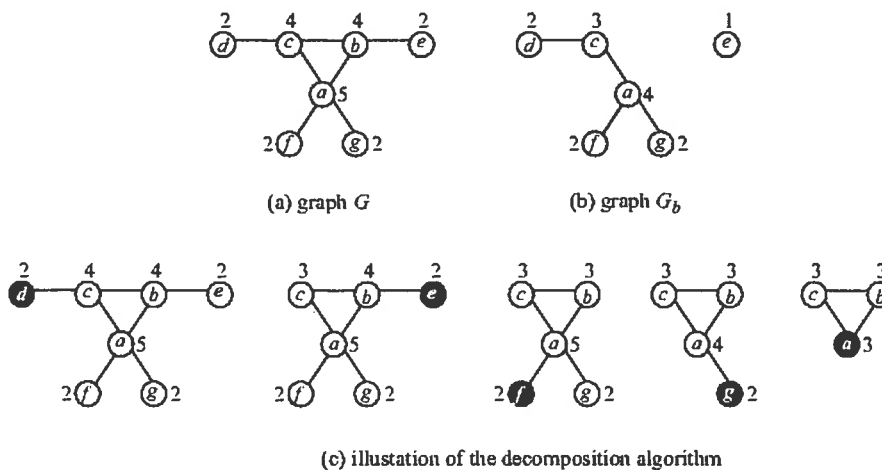


Figure 2. Illustration of the upper bounds

Notice that if $h(G) = |V(G)|$, then the decomposition algorithm always chooses a vertex with minimum degree in the remaining graph. Hence, it is equivalent to the procedure proposed by Szekeres and Wilf [SW68] for the computation of an upper bound on the chromatic number $\chi(G)$. For other upper bounding procedures h , our decomposition algorithm possibly gives a bound $h^*(G) < \chi(G)$. For example, assume that h is a procedure that returns the number of colors used by a linear coloring algorithm that orders the vertices randomly and then colors them sequentially according to that order, giving the smallest available color to each vertex. We then have $h(G) \geq \chi(G)$. However, for G equal to a pentagon (the

chordless cycle on five vertices), $h(N_G(u)) = 2$ for all $u \in V(G)$, which implies $\chi(G) = 3 > 2 = h^*(G) = h^1(G) = h_s^2(G)$ for all $s \in V(G)$.

The computational complexity of the decomposition algorithm is $O(n^2 T(n))$, where $T(n)$ is the time taken to compute $h(G)$ on a graph G of size n . Since $h^1(G)$ and $h_s^2(G)$ can both be computed in $O(n T(n))$, significant improvements in the quality of the upper bounds need to be observed to justify this additional computational effort. Our computational results, presented in Section 2.6.1, show that this is indeed the case. Before presenting these results, we will see how to use the decomposition algorithm to also improve the computation of lower bounds on the clique number of a graph.

2.5 Using the Decomposition Algorithm to Compute Lower Bounds

In this section we show how to exploit the results of the decomposition algorithm to compute lower bounds on the clique number of a graph. To this end, we make use of the following proposition.

Proposition 5. *Let $h^*(G)$ and G^* be the output of the decomposition algorithm. If $h^*(G) = \omega(G)$, then G^* contains all cliques of maximum size of G .*

Proof. Suppose there exists a clique of size $\omega(G)$ in G that is not in G^* , and let t^* be the iteration where $h^*(G)$ was updated last. At some iteration t' , prior to t^* , some vertex s' belonging to a maximum clique of G was removed from some graph $G' \subseteq G$. Hence $h^*(G) \geq h(N_{G'}(s')) \geq \omega(N_{G'}(s')) = \omega(G)$ at the end of iteration t' . Since $h^*(G)$ is updated (increased) at iteration t^* , we have $h^*(G) > \omega(G)$ at the end of iteration t^* . \square

Notice that when $h^*(G) > \omega(G)$, it may happen that $\omega(G^*) < \omega(G)$. For example, for the left graph G of Figure 3, with $h(G) = |V(G)|$, the decomposition algorithm first selects $s = a$ or b , say a , and $h^*(G)$ is set equal to 3. Then b is removed without changing the value of $h^*(G)$. Finally, one of the 6 remaining vertices is selected, the bound $h^*(G)$ is set equal to 4, and the algorithm stops since

there are no vertices with $h(N_{G'}(u)) > 4$ in the remaining graph G' . Hence G^* has 6 vertices and $\omega(G^*) = 2 < 3 = \omega(G)$. According to Proposition 5, this is possible only because $h^*(G) = 4 > 3 = \omega(G)$.

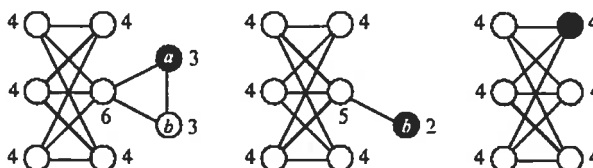


Figure 3. A graph G with $\omega(G^*) < \omega(G)$

In order to obtain a lower bound on $\omega(G)$, Proposition 5 suggests to determine G^* , and to run an algorithm (either exact or heuristic) to get a lower bound on the clique number of G^* . Clearly, the size of such a clique is a lower bound on the clique number of G . This process is summarized in Figure 4, where $\ell(G)$ is any known procedure that computes a lower bound on the clique number of a graph G , while $\ell^*(G)$ is the new lower bound produced by our algorithm.

Lower bounding procedure

1. Run the decomposition algorithm on G to get G^* ;
2. Set $\ell^*(G) \leftarrow \ell(G^*)$.

Figure 4. Algorithm for the computation of a lower bound on $\omega(G)$

When G^* is small enough, an exact algorithm can be used at Step 2 to determine $\omega(G^*)$, while it can be too time consuming to use the same exact algorithm to compute $\omega(G)$. However, for many instances, the iteration where $h^*(G)$ is updated last is reached very early, as will be shown in the next section, and using an exact algorithm at Step 2 is often not realistic. We will observe in the next section that even if ℓ is a heuristic lower bounding function, it often happens that $\ell^*(G) = \ell(G^*) > \ell(G)$. Notice that such a situation can only happen if ℓ is not a decreasing

function (i.e., $\ell(G')$ is possibly larger than $\ell(G)$ for a subgraph $G' \subset G$). This is illustrated with ℓ equal to the well-known greedy algorithm MIN [HRS02] described in Figure 5; we then use the notation $\ell(G) = \text{MIN}(G)$.

Procedure MIN

```

Set  $G' \leftarrow G$  and  $K \leftarrow \emptyset$ ;
While  $G' \neq \emptyset$  do
  Set  $s \leftarrow \operatorname{argmax}_{u \in V(G') \setminus K} |N_{G'}(u)|$ ;
  Set  $G' \leftarrow N_{G'}(s)$  and  $K \leftarrow K \cup \{s\}$ ;
Return  $|K|$ .

```

Figure 5. Greedy lower bounding algorithm for the clique number

Algorithm MIN applied on the graph G of Figure 6 returns value 2, since vertex a has the largest number of neighbors and is therefore chosen first. The decomposition algorithm with $h(G) = |V(G)|$ first chooses $s = b, c$ or d , say b , which gives value 2 to $h^*(G)$. Then c, d and a are removed without changing the value of $h^*(G)$. Finally, one of the vertices e, f or g is selected, which gives $h^*(G) = 3$, and the algorithm stops. Hence, G^* is the triangle induced by vertices e, f and g , and procedure MIN applied to this triangle returns value 3. In summary, we have $\ell^*(G) = \ell(G^*) = 3 > 2 = \ell(G)$. Notice also that even if MIN and h are not very efficient lower and upper bounding procedures (since $\ell(G) = 2 < 3 = \omega(G) < 6 = h(G)$), they help getting better bounds. In our example, we have $\ell^*(G) = \ell(G^*) = 3 = h^*(G)$, which provides a proof that $\omega(G) = 3$.

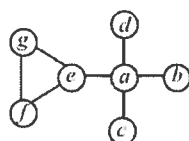


Figure 6. A graph G with $\ell^*(G) > \ell(G)$

2.6 Computational Results

The objective of our computational experiments is twofold. First, we analyze the effectiveness of the decomposition algorithm when using different upper bound functions h . Second, we present the lower bounds obtained by running several maximum clique algorithms (exact and heuristic) on the graph G^* resulting from the decomposition algorithm (using an effective upper bound function h). All our tests were performed on 93 instances used in the second DIMACS challenge [JT93]. Most instances come from the maximum clique section, though we also included a few instances from the graph coloring section, since we use graph coloring heuristics as upper bound functions. The characteristics of the selected instances can be found in the next subsection.

2.6.1 Computing Upper Bounds

Since the decomposition method produces a bound $h^*(G)$ with a significant increase in computing time when compared to the computation of $h(G)$, we did not use hard-to-compute upper bound functions like Lovasz' theta function [Knu94, Lov79] (which gives a value between the clique number and the chromatic number). Although there is a polynomial time algorithm to compute this bound, it is still time-consuming (even for relatively small instances) and difficult to code.

The most trivial bounds we tested are the number of vertices $h_a(G) = |V(G)|$ and the size of the largest closed neighborhood $h_b(G) = \max_{u \in V(G)} |V(N_G(u))|$. Apart from these loose bounds, we obtained tighter bounds by computing upper bounds on the chromatic number with three fast heuristic methods. The simplest is the linear coloring algorithm (which we denote $h_c(G)$), which consists in assigning the smallest available color to each vertex, using the order given in the file defining the graph. The second graph coloring method we tested is DSATUR [Bré79] (denoted $h_d(G)$), a well-known greedy algorithm which iteratively selects a vertex with maximum saturation degree and assigns to it the smallest available color, the saturation degree of a vertex u being defined as the number of colors already

used in $N_G(u)$. Finally, the last method we tested (denoted $h_e(G)$) starts with the solution found by DSATUR and runs the well-known tabu search algorithm Tabucol [HdW87, GH06], performing as many iterations as there are vertices in the graph (which is a very small amount of iterations for a tabu search).

Tables 2.1 and 2.2 give the detailed results obtained when using the decomposition algorithm with these five upper bound functions. The first column (Problem) indicates the name of the problem instance taken from the DIMACS site; the second column (W) gives the size of the largest known clique; the remaining columns indicate the upper bounds $h_x(G)$ and $h_x^*(G)$ computed by each of the five functions on the original graph and when using the decomposition algorithm (where x is any letter between a and e).

To analyze these results, we use the following *improvement ratio*, which is a value in the interval $[0,1]$:

$$I_x = \frac{h_x(G) - h_x^*(G)}{h_x(G) - W}.$$

A value of 0 indicates that the bound $h_x^*(G)$ does not improve upon $h_x(G)$, while a value of 1 corresponds to the case where $h_x^*(G) = W$, i.e., the maximum possible improvement (if W is indeed the clique number) is achieved by the decomposition algorithm. We have discarded the cases where the upper bound function applied to G already found a maximum clique, since then there is no possible improvement to be gained by using the decomposition algorithm. Table 2.3 displays the improvement ratios (in %) averaged for each family of graphs and for all instances. The first column (Problem) indicates the family of graphs, each being represented by the first characters identifying them, followed by a star (*). The remaining columns show the average improvement for the five upper bound functions.

Our initial conjecture was that the improvement would be inversely correlated to the quality of the upper bound function, i.e., the worst the function, the more room for improvement there is, hence the most improvement should be obtained. The results do not verify this conjecture, since the best upper bound functions show better improvements than the worst ones. Indeed, the worst functions, $h_a(G)$

Problem	W	$h_a(G)$	$h_a^*(G)$	$h_b(G)$	$h_b^*(G)$	$h_c(G)$	$h_c^*(G)$	$h_d(G)$	$h_d^*(G)$	$h_e(G)$	$h_e^*(G)$
brock200_1	21	200	135	166	114	59	42	53	38	47	35
brock200_2	12	200	85	115	54	36	19	31	17	30	16
brock200_3	15	200	106	135	76	45	28	39	24	35	23
brock200_4	17	200	118	148	90	49	32	43	29	41	27
brock400_1	27	400	278	321	226	102	75	93	68	89	62
brock400_2	29	400	279	329	229	100	74	93	68	90	62
brock400_3	31	400	279	323	227	103	75	92	68	83	63
brock400_4	33	400	278	327	228	100	74	91	68	90	62
brock800_1	23	800	488	561	345	144	96	137	88	128	80
brock800_2	24	800	487	567	347	144	96	134	88	122	81
brock800_3	25	800	484	559	346	143	96	133	87	123	80
brock800_4	26	800	486	566	346	148	96	136	87	124	81
c-fat200-1	12	200	15	18	15	12	12	15	12	14	12
c-fat200-2	24	200	33	35	33	24	24	24	24	24	24
c-fat200-5	58	200	84	87	84	68	58	84	58	83	58
c-fat500-1	14	500	18	21	18	14	14	14	14	14	14
c-fat500-10	126	500	186	189	186	126	126	126	126	126	126
c-fat500-2	26	500	36	39	36	26	26	26	26	26	26
c-fat500-5	64	500	93	96	93	64	64	64	64	64	64
c1000_9	68	1000	875	926	814	319	283	305	266	276	238
c125_9	34	125	103	120	100	57	49	52	44	47	41
c2000_5	16	2000	941	1075	518	226	120	210	110	198	102
c2000_9	77	2000	1759	1849	1625	592	519	562	492	492	442
c250_9	44	250	211	237	200	98	86	92	78	82	71
c4000_5	18	4000	1910	2124	1028	402	215	377	200	365	186
c500_9	57	500	433	469	408	184	156	164	144	149	131
dsjc1000_5	15	1000	460	552	263	127	68	115	61	109	57
dsjc500_5	13	500	226	287	134	72	39	65	35	61	33
gen200_p0_9_44	44	200	168	191	161	76	63	62	53	44	44
gen200_p0_9_55	55	200	167	191	161	80	68	71	60	62	55
gen400_p0_9_55	55	400	337	376	320	127	110	102	81	55	55
gen400_p0_9_65	65	400	337	379	321	136	118	118	99	65	65
gen400_p0_9_75	75	400	337	381	322	143	124	118	103	75	79
hamming10-2	512	1024	1014	1014	1006	512	512	512	512	512	512
hamming10-4	40	1024	849	849	724	128	121	85	71	85	70
hamming6-2	32	64	58	58	54	32	32	32	32	32	32
hamming6-4	4	64	23	23	8	8	5	7	5	7	5
hamming8-2	128	256	248	248	242	128	128	128	128	128	128
hamming8-4	16	256	164	164	110	32	27	24	17	24	16
johnson16-2-4	8	120	92	92	68	14	13	14	13	14	13
johnson32-2-4	16	496	436	436	380	30	29	30	29	30	29
johnson8-2-4	4	28	16	16	8	6	5	6	5	6	5
johnson8-4-4	14	70	54	54	42	20	17	17	14	14	14
keller4	11	171	103	125	76	37	18	24	17	22	15
keller5	27	776	561	639	459	175	50	61	49	59	43
keller6	59	3361	2691	2953	2350	781	126	141	122	141	118

Table 2.1: Upper bounds obtained with five upper bound functions $h_x(G)$

Problem	W	$h_a(G)$	$h_a^*(G)$	$h_b(G)$	$h_b^*(G)$	$h_c(G)$	$h_c^*(G)$	$h_d(G)$	$h_d^*(G)$	$h_e(G)$	$h_e^*(G)$
latin_square_10.col	90	900	684	684	636	213	144	132	108	119	105
le450_15a.col	15	450	25	100	18	22	15	17	15	17	15
le450_15b.col	15	450	25	95	17	22	15	16	15	16	15
le450_15c.col	15	450	50	140	29	30	15	23	15	23	15
le450_15d.col	15	450	52	139	29	31	15	24	15	23	15
le450_25a.col	25	450	27	129	26	28	25	25	25	25	25
le450_25b.col	25	450	26	112	25	27	25	25	25	25	25
le450_25c.col	25	450	53	180	39	37	25	29	25	29	25
le450_25d.col	25	450	52	158	38	35	25	28	25	28	25
le450_5a.col	5	450	18	43	7	14	5	10	5	10	5
le450_5b.col	5	450	18	43	7	13	5	9	5	9	5
le450_5c.col	5	450	34	67	12	17	6	10	5	9	5
le450_5d.col	5	450	33	69	12	18	5	12	5	11	5
MANN_a27	126	378	365	375	363	135	135	140	137	135	131
MANN_a45	345	1035	1013	1032	1011	372	370	369	367	363	353
MANN_a81	1100	3321	3281	3318	3279	1134	1134	1153	1146	1135	1124
MANN_a9	16	45	41	42	39	18	18	19	18	18	17
p_hat1000-1	10	1000	164	409	84	69	24	52	20	52	19
p_hat1000-2	46	1000	328	767	289	148	89	109	76	109	74
p_hat1000-3	68	1000	610	896	554	230	160	187	134	187	132
p_hat1500-1	12	1500	253	615	126	95	33	74	28	74	27
p_hat1500-2	65	1500	505	1154	451	213	133	157	113	157	112
p_hat1500-3	94	1500	930	1331	840	326	231	270	195	270	194
p_hat300-1	8	300	50	133	28	29	11	22	9	21	9
p_hat300-2	25	300	99	230	90	56	34	42	29	42	28
p_hat300-3	36	300	181	268	166	85	59	69	51	69	49
p_hat500-1	9	500	87	205	46	45	16	32	13	32	13
p_hat500-2	36	500	171	390	152	87	54	66	46	66	45
p_hat500-3	50	500	304	453	279	131	94	108	78	107	76
p_hat700-1	11	700	118	287	62	53	19	40	16	40	16
p_hat700-2	44	700	236	540	213	114	71	85	60	85	59
p_hat700-3	62	700	427	628	389	171	120	143	102	141	100
san1000	15	1000	465	551	400	47	21	24	16	15	15
san200_0_7_1	30	200	126	156	108	49	32	42	30	31	30
san200_0_7_2	18	200	123	165	113	35	24	23	18	18	18
san200_0_9_1	70	200	163	192	156	92	75	73	70	70	70
san200_0_9_2	60	200	170	189	161	86	75	75	63	62	60
san200_0_9_3	44	200	170	188	161	73	65	64	53	48	44
san400_0_5_1	13	400	184	226	155	29	14	21	13	21	13
san400_0_7_1	40	400	262	302	224	81	54	59	43	45	40
san400_0_7_2	30	400	260	305	217	67	49	47	36	30	30
san400_0_7_3	22	400	254	308	203	59	42	29	26	22	22
san400_0_9_1	100	400	345	375	325	163	138	135	116	109	100
sanr200_0_7	18	200	125	162	100	52	36	47	32	42	30
sanr200_0_9	42	200	167	190	158	82	70	74	64	69	59
sanr400_0_5	13	400	178	234	108	62	33	56	29	50	27
sanr400_0_7	21	400	259	311	201	91	64	83	58	78	53

Table 2.2: Upper bounds obtained with five upper bound functions $h_x(G)$

Problem	$I_a(\%)$	$I_b(\%)$	$I_c(\%)$	$I_d(\%)$	$I_e(\%)$
brock*	41	40	45	47	50
c-fat*	93	23	100	100	100
c*	27	27	30	32	33
dsjc*	56	54	54	56	57
gen*	20	20	33	47	100
hamming*	25	25	38	62	67
johnson*	29	35	31	43	25
keller*	30	31	83	37	47
latin*	27	8	56	57	48
le*	96	93	99	100	100
MANN*	6	5	2	19	45
p-hat*	66	63	62	64	66
san*	36	29	61	79	100
sanr*	39	40	44	47	48
ALL	50	43	55	60	64

Table 2.3: Average improvement for each family of graphs

and $h_b(G)$, display improvements of 50% and 43%, respectively, while the best functions, $h_c(G)$, $h_d(G)$ and $h_e(G)$, reach improvements of 55%, 60% and 64%, respectively. Even among the graph coloring algorithms, we observe an inverse relationship, i.e., as the effectiveness of the method increases, the improvement values also increase. At first, we thought this phenomenon might be due to the fact that for some families of graphs average improvements were reaching 100%, thus influencing the overall average improvement more than it should. But a similar progression can be observed for most families of graphs. Another explanation would be that if two functions show different results when applied to G but identical results when embedded into the decomposition algorithm, then the improvement would be better for the best function because the denominator is smaller. When we look at the detailed results, however, we notice that in general, the better the function, the better the results obtained by the decomposition algorithm.

2.6.2 Computing Lower Bounds

In this section, we present the results obtained when computing lower bounds using G^* , the graph obtained at the iteration where $h^*(G)$ was updated last in the decomposition algorithm. We use DSATUR ($h_d(G)$) as upper bound function, since it shows a good balance between solution effectiveness and computational

efficiency. We tested four maximum clique algorithms to compute lower bounds:

- An exact branch-and-bound algorithm, dfmax [JT93] (available as a C program on the DIMACS ftp site [AJ05]), performed with a time limit of five hours. We denote the lower bound obtained by this algorithm when applied to G and G^* as $l_a(G)$ and $l_a(G^*)$, respectively.
- A very fast greedy heuristic, MIN [HRS02] (already described in Section 2.5). We denote the lower bounds obtained by this algorithm when applied to G and G^* as $l_b(G)$ and $l_b(G^*)$, respectively.
- The penalty-evaporation heuristic [SL02], which, at each iteration, inserts into the current clique some vertex i , removing the vertices not adjacent to i . The removed vertices are penalized in order to reduce their potential of being selected to be inserted again during the next iterations. This penalty is gradually evaporating. We denote the lower bounds obtained by this algorithm when applied to G and G^* as $l_c(G)$ and $l_c(G^*)$, respectively.
- An improved version of the above penalty-evaporation heuristic, summarized in Figure 7. We denote the lower bounds obtained by this algorithm when applied to G and G^* as $l_d(G)$ and $l_d(G^*)$, respectively.

Improved penalty-evaporation heuristic

1. Set $IPE(G) \leftarrow 0$ and $G' \leftarrow G$;
2. Run the penalty-evaporation heuristic on G' to get a clique K ;
If $|K| > IPE(G)$ then set $IPE(G) \leftarrow |K|$;
3. For all $u \in K$ do
 - Run the penalty-evaporation heuristic on $N_{G'}(u)$ to get a clique K' ;
 - If $|K'| > IPE(G)$ then set $IPE(G) \leftarrow |K'|$, $K \leftarrow K'$, and restart Step 3;
4. Remove K from G' (i.e., set G' equal to the subgraph induced by $V(G') \setminus K$);

If G' is not empty then go to 2 else STOP: return $IPE(G)$.

Figure 7. Improved penalty-evaporation heuristic [SL02]

The results obtained on the same instances as in Section 2.6.1 are presented in Tables 2.4 and 2.5. Of the 93 instances, we removed those where G and G^* coincide, which left 72 instances. The first column (Problem) indicates the name of each problem; the second and third columns show the number of vertices in G (n) and G^* (n^*), respectively; the fourth column (W) gives the size of the largest known clique; the fifth and sixth columns ($l_a(G)$ and $l_a(G^*)$) show the results obtained by dfmax (with a time limit of five hours) when applied to G and G^* , respectively (a + sign indicates the algorithm was stopped because of the time limit, so G or G^* might contain a clique of size larger than the given value); the remaining columns give the lower bounds generated by the three maximum clique heuristic methods, when applied to G and G^* .

Problem	n	n^*	W	$l_a(G)$	$l_a(G^*)$	$l_b(G)$	$l_b(G^*)$	$l_c(G)$	$l_c(G^*)$	$l_d(G)$	$l_d(G^*)$
brock200_1	200	198	21	21	21	14	16	20	20	20	21
brock200_2	200	197	12	12	12	7	8	10	11	11	12
brock200_3	200	199	15	15	15	10	11	14	13	14	14
brock200_4	200	195	17	17	17	11	13	16	16	17	17
brock400_1	400	399	27	27+	27+	19	19	23	23	25	24
brock400_2	400	399	29	29+	29+	20	18	23	24	29	25
brock400_3	400	399	31	31+	31+	20	17	24	25	31	31
brock800_3	800	799	25	21+	21+	15	15	19	20	22	22
c-fat200-1	200	90	12	12	12	12	12	12	12	12	12
c-fat200-2	200	24	24	24	24	24	24	24	24	24	24
c-fat200-5	200	116	58	58	58	58	58	58	58	58	58
c-fat500-1	500	140	14	14	14	14	14	14	14	14	14
c-fat500-10	500	252	126	124+	126	126	126	126	126	126	126
c-fat500-2	500	260	26	26	26	26	26	26	26	26	26
c-fat500-5	500	128	64	64	64	64	64	64	64	64	64
c1000_9	1000	997	68	53+	52+	51	51	64	64	67	67
c2000_5	2000	1999	16	16+	16+	10	10	15	15	16	16
c250_9	250	242	44	41+	42+	35	36	44	44	44	44
c4000_5	4000	3998	18	17+	17+	12	12	16	17	18	17
c500_9	500	498	57	47+	47+	42	47	56	54	57	57
dsjc1000_5	1000	998	15	15	15	10	10	14	14	15	15
dsjc500_5	500	498	13	13	13	10	10	12	12	13	13
gen200_p0_9_44	200	197	44	44+	44+	32	32	44	44	44	44
gen200_p0_9_55	200	196	55	55	55	36	37	55	55	55	55
gen400_p0_9_55	400	388	55	43+	44+	42	44	51	51	53	52
gen400_p0_9_65	400	398	65	43+	43+	40	40	65	52	65	65
gen400_p0_9_75	400	398	75	45+	45+	45	47	75	75	75	75
hamming10-4	1024	1023	40	32+	34+	29	27	40	40	40	40
hamming8-4	256	114	16	16	16	16	11	16	16	16	16
keller5	776	770	27	24+	25+	15	15	26	27	27	27
keller6	3361	3338	59	42+	45+	32	36	39	43	59	59

Table 2.4: Lower bounds $l_x(G)$ and $l_x(G^*)$

Column $l_a(G^*)$ indicates that dfmax has determined $\omega(G^*)$ within the time limit of five hours for 41 out the 72 instances. By comparing columns W , $l_a(G)$

Problem	n	n^*	W	$l_a(G)$	$l_a(G^*)$	$l_b(G)$	$l_b(G^*)$	$l_c(G)$	$l_c(G^*)$	$l_d(G)$	$l_d(G^*)$
latin_square_10.col	900	88	90	90+	81+	90	90	90	90	90	90
le450_15a.col	450	335	15	15	15	5	6	15	15	15	15
le450_15b.col	450	341	15	15	15	8	8	15	15	15	15
le450_15c.col	450	430	15	15	15	7	7	15	15	15	15
le450_15d.col	450	434	15	15	15	5	5	15	15	15	15
le450_25a.col	450	217	25	25	25	11	9	25	25	25	25
le450_25b.col	450	237	25	25	25	13	12	25	25	25	25
le450_25c.col	450	376	25	25	25	7	8	25	25	25	25
le450_25d.col	450	373	25	25	25	8	9	25	25	25	25
le450_5a.col	450	392	5	5	5	4	5	5	5	5	5
le450_5b.col	450	388	5	5	5	4	3	5	5	5	5
le450_5c.col	450	449	5	5	5	3	3	5	5	5	5
p_hat1000-1	1000	665	10	10	10	7	9	10	10	10	10
p_hat1000-2	1000	470	46	43+	41+	38	40	46	46	46	46
p_hat1000-3	1000	853	68	49+	48+	57	57	67	68	68	68
p_hat1500-1	1500	752	12	12	11	8	7	12	11	12	11
p_hat1500-2	1500	690	65	46+	48+	54	59	65	65	65	65
p_hat1500-3	1500	1263	94	53+	54+	75	81	94	94	94	94
p_hat300-1	300	245	8	8	8	7	7	8	8	8	8
p_hat300-2	300	169	25	25	25	23	20	25	25	25	25
p_hat300-3	300	279	36	36	36	31	31	36	36	36	36
p_hat500-1	500	372	9	9	9	6	8	9	9	9	9
p_hat500-2	500	257	36	36	36	29	33	36	36	36	36
p_hat500-3	500	448	50	44+	48+	42	42	49	50	50	50
p_hat700-1	700	487	11	11	11	7	7	11	11	11	11
p_hat700-2	700	324	44	44	44	38	42	44	44	44	44
p_hat700-3	700	621	62	50+	51+	55	53	62	62	62	62
san1000	1000	970	15	10+	10+	8	8	8	8	15	10
san200_0_7_1	200	30	30	30	30	16	30	17	30	30	30
san200_0_7_2	200	189	18	18+	18+	12	12	13	13	18	18
san200_0_9_1	200	70	70	48+	70	45	70	45	70	70	70
san200_0_9_3	200	198	44	44+	36+	31	30	36	43	44	43
san400_0_5_1	400	13	13	13	13	7	13	8	13	13	13
san400_0_7_1	400	390	40	22+	20+	21	20	21	20	22	22
san400_0_7_2	400	398	30	17+	17+	15	15	18	18	30	30
san400_0_7_3	400	376	22	17+	22+	12	12	17	17	22	22
san400_0_9_1	400	393	100	49+	50+	41	39	54	100	100	100
sanr200_0_7	200	197	18	18	18	14	14	18	18	18	18
sanr200_0_9	200	198	42	40+	40+	35	36	42	42	42	42
sanr400_0_5	400	395	13	13	13	10	9	13	12	13	13
sanr400_0_7	400	397	21	21	21	16	16	21	20	21	21

Table 2.5: Lower bounds $l_x(G)$ and $l_x(G^*)$

and $l_a(G^*)$ on these instances, we observe that $\omega(G^*) = \omega(G) = W$ in 38 cases, while $\omega(G^*) < \omega(G)$ in one case (instance p_hat1500-1) and $\omega(G^*) = W$ and $\omega(G)$ is not known in two cases (instances c-fat500-10 and san200_0_9_1). We do not report computing times since the aim of the experiments is to compare the quality of the lower bounds on G and G^* . We find however interesting to mention that for 22 of the 38 instances with $\omega(G^*) = \omega(G)$, dfmax has determined the clique number, both in G and G^* , in less than 1 second. For the 16 other instances, the decrease in computing time is in average equal to 47%.

For the 31 instances for which $\omega(G^*)$ is not known, we deduce from columns W , $l_a(G^*)$ and $l_d(G^*)$ that $\omega(G^*) \geq W$ in 24 cases. The status of the seven remaining instances is yet unknown. Also, $l_a(G) < l_a(G^*)$ for 11 out of these 31 instances while $l_a(G) > l_a(G^*)$ for 6 of them (and $l_a(G) = l_a(G^*)$ for the 14 remaining instances).

Furthermore, let $\frac{|V(G)| - |V(G^*)|}{|V(G)|}$ be the reduction ratio between the number of vertices in graphs G and G^* . The mean reduction ratio for the 72 instances is 20%, among which 36 instances have a reduction ratio of more than 5%. If we focus on these 36 instances, we find 35 instances with $\omega(G^*) \geq W$ and one (p-hat1500-1) with $\omega(G^*) < W$.

In general, it seems preferable to perform a heuristic method on G^* rather than on G . When MIN (l_b) is used, there are 25 instances with better results on G^* and only 14 instances with better results on G (for the other instances, we obtain the same results on G and G^*). Similarly, the penalty-evaporation method (l_c) is better when applied to G^* in 14 cases, and better when applied to G for 7 instances. For the last method, the situation is reversed, since $l_d(G^*) > l_d(G)$ in only two cases, while $l_d(G^*) < l_d(G)$ for 7 instances. For one of these 7 instances, we know that $\omega(G^*) < \omega(G)$, while for four other instances, we do not know whether G^* contains a maximum clique of G or not.

2.7 Conclusion

In this paper, we have presented a decomposition algorithm to compute an upper bound on the clique number of a graph. At each iteration, this algorithm removes one vertex and updates a tentative upper bound by considering the closed neighborhoods of the remaining vertices. Given any method to compute an upper bound on the clique number of a graph, we have shown, under mild assumptions, that the decomposition algorithm is guaranteed to improve upon that upper bound. Our computational results on DIMACS instances show significant improvements of about 60%. We have also shown how to use the output of the decomposition

algorithm to improve the computation of lower bounds.

It would be interesting to apply the decomposition algorithm to other upper bound functions to see if similar trends can be observed. The development of other heuristic methods based on the decomposition algorithm is another promising avenue for future research.

CHAPITRE 3

ÉTUDE DE VOISINAGES BASÉS SUR L'ORIENTATION D'ARÊTES POUR LE PROBLÈME DE LA COLORATION MINIMUM

La première référence utile faisant le lien entre les chemins dans un graphe et le nombre de couleurs requises pour colorer les noeuds du graphe provient d'un théorème de Gallai [Gal68], Roy [Roy67], et Vitaver [Vit62]. Le théorème implique que si l'on donne une orientation aux arêtes d'un graphe, alors le nombre de noeuds dans le plus long chemin du graphe est au moins égal au nombre chromatique. On peut ensuite en extraire une corrélation entre les orientations sans circuits d'un graphe et les colorations de ce même graphe. Pour ce faire, il suffit d'associer une relation d'ordre aux arcs d'un graphe ; par exemple, si (i, j) est un arc dans le graphe, alors le noeud i a une couleur plus petite que celle du noeud j . La relation inverse est également utilisable (comme c'est le cas dans l'article de Barbosa et al. [BAdN04]). Pour obtenir une orientation sans circuits à partir d'une coloration, il suffit d'orienter toutes les arêtes (i, j) de i vers j si le noeud i possède une couleur plus petite que j , et l'inverse sinon. Pour obtenir une coloration à partir d'une orientation sans circuits, il suffit de donner la couleur 1 à tous les noeuds qui n'ont pas de prédécesseurs, et ensuite de colorer itérativement les noeuds dont tous les prédécesseurs sont colorés, en utilisant la couleur la plus petite possible. Cette bijection nous permet de conclure que trouver l'orientation du graphe qui minimise la longueur du plus long chemin correspond à colorer le graphe en utilisant le moins de couleurs possibles.

À notre connaissance, aucun algorithme de recherche locale n'a été développé dans ce contexte. Nous introduisons quatre voisinages, que nous avons intégrés à un algorithme tabou pour en évaluer l'efficacité relative les uns par rapport aux autres, ainsi que des propriétés concernant les mouvements utilisés dans la définition de ces voisinages. Nous nous sommes intéressés à deux types de mouvements : l'inversion

d'un ou plusieurs arcs pour tenter de briser les plus longs chemins, et l'ajout et le retrait d'arcs dans le graphe pour tenter d'obtenir une solution dont les plus longs chemins ne dépassent pas une limite prédéterminée. Les prochaines sections décrivent plus en détail ces voisinages.

3.1 Voisinages

Pour les trois prochaines sections, nous considérons une orientation sans circuits d'un graphe quelconque et nous identifions des propriétés associées à l'inversion d'un ou plusieurs arcs. En particulier, nous tentons de savoir si le graphe résultant possède des chemins plus longs ou plus courts, et si des circuits se sont formés. De plus, nous tâchons de déterminer s'il existe une suite d'inversions permettant de passer d'une orientation sans circuits quelconque à une orientation optimale.

Pour la quatrième section, nous considérons un graphe partiellement orienté, où les arêtes qui ne sont pas orientées sont retirées du graphe et où la longueur du plus long chemin ne dépasse pas une constante k . Nous identifions des propriétés associées à l'insertion d'un arc et au retrait d'un ou plusieurs arcs. En particulier, nous déterminons si le graphe résultant possède des chemins plus longs que k , et si des circuits se sont formés. De plus, nous tâchons de déterminer s'il existe une suite d'insertions et de retraits permettant de passer d'une orientation partielle sans circuits dont la taille des plus longs chemins ne dépasse pas k , à une orientation complète dont les plus longs chemins ne dépassent toujours pas k .

Dans tous les cas, nous référons le lecteur à l'article pour tous les détails des preuves et démonstrations.

3.1.1 Inversion d'un arc

Si l'on inverse un arc (i, j) faisant partie d'un chemin de longueur maximale dans le graphe, alors, puisque le graphe n'a pas de circuits avant l'inversion, il n'en aura pas non plus après. De plus, la longueur du plus long chemin ne peut augmenter que d'au plus 2. Enfin, si inverser cet arc augmente la longueur du plus long chemin,

alors il existe nécessairement un autre arc sur un chemin de longueur maximale dans le graphe original qui avait la même origine (i) ou la même destination (j).

Nous appellerons mouvement *ArcSimple*, le mouvement consistant à choisir un arc sur un plus long chemin et à l'inverser. Nous avons démontré qu'il est possible d'atteindre une solution optimale à partir d'une solution quelconque en effectuant itérativement un mouvement de type *ArcSimple*. Le voisinage appelé *VoisinArcSimple* consiste simplement en l'ensemble des mouvements de type *ArcSimple* qui sont possibles à partir d'une solution quelconque.

3.1.2 Inversion de plusieurs arcs reliés à un noeud

Si l'on inverse tous les arcs sur des plus longs chemins sortant (ou entrant, exclusivement) d'un noeud v , alors puisque le graphe n'a pas de circuits avant l'inversion, il n'en aura pas non plus après. De plus, la longueur du plus long chemin ne peut augmenter que d'au plus 1.

Nous appellerons mouvement *NoeudSimple*, le mouvement consistant à choisir un noeud sur un chemin de longueur maximale et à inverser tous ses arcs sortants ou entrants qui sont sur des chemins de longueur maximale. Nous avons démontré qu'il est possible d'atteindre une solution optimale à partir d'une solution quelconque en effectuant itérativement un mouvement de type *NoeudSimple*. Le voisinage appelé *VoisinNoeudSimple* consiste simplement en l'ensemble des mouvements de type *NoeudSimple* qui sont possibles à partir d'une solution quelconque.

3.1.3 Inversion de plusieurs arcs parmi des composantes connexes

Une composante connexe est un ensemble de noeuds qui sont tous reliés entre eux par au moins un chemin, peu importe l'orientation des arcs sur ce chemin. Un chemin est une séquence de noeuds tels que le premier noeud est relié au second par un arc, le dernier noeud est relié à l'avant-dernier par un arc, et tous les autres noeuds de la séquence sont reliés par un arc au noeud qui les précède et qui les succède dans la séquence.

Considérons tous les noeuds du graphe dont le plus long chemin menant à ces noeuds est de taille l ou $l + 1$, pour une constante $l \geq 0$ quelconque. Identifions les composantes connexes parmi ces noeuds formées uniquement d'arcs faisant partie des plus longs chemins du graphe. Si l'on sélectionne un sous-ensemble de ces composantes connexes, et que l'on inverse tous les arcs en faisant partie, alors puisque le graphe n'a pas de circuits avant l'inversion, il n'en aura pas non plus après. De plus, la longueur du plus long chemin ne peut pas augmenter.

Nous appellerons mouvement *CompConn*, le mouvement consistant à choisir un $l \geq 0$, à identifier un sous-ensemble de composantes connexes formées d'arcs faisant partie des plus longs chemins, où le plus long chemin menant aux noeuds des composantes connexes est l ou $l + 1$, et à en inverser tous les arcs. Nous avons trouvé un exemple de graphe avec une solution initiale telle qu'il est impossible d'atteindre une solution optimale en effectuant itérativement un mouvement de type *CompConn*. Le voisinage appelé *VoisinCompConn* consiste simplement en l'ensemble des mouvements de type *CompConn* qui sont possibles à partir d'une solution quelconque.

3.1.4 Ajout et retrait d'arcs

Si l'on ajoute un arc (i, j) dans le graphe, où le plus long chemin entrant dans i est de longueur l et le plus long chemin entrant dans j est de longueur strictement supérieure à l , alors puisque le graphe n'a pas de circuits avant l'insertion, il n'en aura pas non plus après. De plus, la longueur du plus long chemin ne peut pas augmenter. Par contre, si le plus long chemin entrant dans j n'est pas strictement supérieur à l , alors si le plus long chemin sortant de j est de longueur l' , la longueur du plus long chemin du graphe ne peut augmenter que si $l + l'$ est supérieur à la taille du plus long chemin du graphe. Si l'on retire un ensemble d'arcs dans le graphe, alors la longueur du plus long chemin diminue si et seulement si tous les chemins de longueur maximale passent par au moins un des arcs retirés.

Nous appellerons mouvement *AjoutRetrait*, le mouvement consistant à choisir une arête (i, j) qui ne fait pas partie du graphe partiel, à lui donner une orienta-

tion, à l'insérer dans le graphe partiel et à retirer un ensemble minimal d'arcs de telle sorte que le graphe est sans circuits et que la longueur du plus long chemin ne dépasse pas k , la longueur du plus long chemin du graphe partiel (avant l'insertion). Nous avons démontré qu'il est possible d'obtenir une orientation complète à partir d'une orientation partielle en limitant la longueur du plus long chemin à une valeur $k \geq \chi(G)$, et en effectuant itérativement un mouvement de type *AjoutRetrait*. Le voisinage appelé *VoisinAjoutRetrait* consiste simplement en l'ensemble des mouvements de type *AjoutRetrait* qui sont possibles à partir d'une solution quelconque.

3.2 Résultats expérimentaux

Nous avons testé de façon sommaire les performances des différents voisinages dans le cadre d'un algorithme de recherche locale de type tabou. Le but de ces tests est d'obtenir une idée de l'efficacité relative des différents types de mouvements. Nous avons comptabilisé des statistiques pour chacun des voisinages afin d'étudier la taille de ces voisinages, le temps de calcul par itération, ainsi que la fréquence d'amélioration et de détérioration de la solution. Le meilleur voisinage semble être *VoisinCompConn*, malgré le fait que ce soit le seul pour lequel l'espace des solutions n'est pas nécessairement connexe. Le deuxième meilleur voisinage est *VoisinAjoutRetrait*, suivi de loin par *VoisinArcSimple* et *VoisinNoeudSimple*.

Ces résultats préliminaires nous indiquent qu'il ne suffit pas d'inverser des arcs sur des chemins de longueur maximale (ou de les retirer) de manière quelconque si l'on veut obtenir de bonnes solutions. Identifier des ensembles connexes semble être une bonne voie, mais peut-être y a-t-il d'autres structures propices à l'inversion d'arcs. Ces voisinages peuvent être utilisés pour faire de la diversification, car un seul mouvement peut changer la coloration d'une grande quantité de noeuds.

ON EDGE ORIENTING METHODS FOR GRAPH COLORING

Bernard Gendron

Département d'informatique et de recherche opérationnelle

Université de Montréal, Canada

gendron@iro.umontreal.ca

Alain Hertz

Département de Mathématiques et de Génie Industriel

École Polytechnique, Montréal, Canada

Alain.Hertz@gerad.ca

Patrick St-Louis

Département d'informatique et de recherche opérationnelle

Université de Montréal, Canada

stlouip@iro.umontreal.ca

Abstract

We consider the problem of orienting the edges of a graph so that the length of a longest path in the resulting digraph is minimum. As shown by Gallai, Roy and Vitaver, this edge orienting problem is equivalent to finding the chromatic number of a graph. We study various properties of edge orienting methods in the context of local search for graph coloring. We then exploit these properties to derive four tabu search algorithms, each based on a different neighborhood. We compare these algorithms numerically to determine which are the most promising and to give potential research directions.

3.3 Introduction

All graphs in this paper have no loops and no multiple edges. For a graph G , we denote $V(G)$ its vertex set and $E(G)$ its edge set. A *partial* graph of G is a graph obtained from G by removing some edges. A k -coloring of G is a function $c : V(G) \rightarrow \{1, \dots, k\}$. It is said *legal* if $c(i) \neq c(j)$ for all edges (i, j) in $E(G)$. The smallest integer k such that a legal k -coloring exists for G is the chromatic number $\chi(G)$ of G . Finding the chromatic number of a given graph is known as the *graph coloring problem*, and is NP-hard [GJ79]. Although many exact algorithms have been devised for this particular problem [Bro72, HH02b, KJ85, MT96, Pee83], such algorithms can only be used to solve small instances.

A directed graph (or just *digraph*) is a graph with an orientation on each edge. An edge (u, v) oriented from u to v is called an *arc*, is denoted $u \rightarrow v$, and u is its *tail* while v is its *head*. An *orientation* of a graph G is a digraph, denoted \vec{G} , obtained from G by choosing an orientation $u \rightarrow v$ or $v \rightarrow u$ for each edge $(u, v) \in E(G)$. By removing some arcs in \vec{G} , one gets a *partial* digraph of \vec{G} . A *path* is a digraph, denoted $(v_0 \rightarrow \dots \rightarrow v_{k-1})$, with vertices v_0, \dots, v_{k-1} and arcs $v_{i-1} \rightarrow v_i$ ($i = 1, \dots, k-1$), and its *length* is its number of vertices. We denote $\lambda(\vec{G})$ the length of a longest path in \vec{G} . Also, we denote $d_{\vec{G}}^+(v)$ the length of a longest path in \vec{G} starting at v , while $d_{\vec{G}}^-(v)$ denotes the length of a longest path ending at v . Hence, $\lambda(\vec{G}) = d_{\vec{G}}^-(u) + d_{\vec{G}}^+(v)$ for every arc $u \rightarrow v$ lying on a longest path in \vec{G} .

A *source* is a vertex v with $d_{\vec{G}}^-(v) = 0$ while a *sink* is a vertex v with $d_{\vec{G}}^+(v) = 0$. A *circuit* is a digraph, denoted $(v_0 \rightarrow \dots \rightarrow v_{k-1} \rightarrow v_0)$, and obtained by adding an arc $v_{k-1} \rightarrow v_0$ to a path $(v_0 \rightarrow \dots \rightarrow v_{k-1})$. A *cycle* is an undirected circuit. A *dicycle* is an orientation of a cycle, which means that every circuit is a dicycle. If a dicycle \vec{C} is not a circuit, then it contains a set $A = \{v_0, \dots, v_{k-1}\}$ of sources and a set $B = \{w_0, \dots, w_{k-1}\}$ of sinks ($k > 0$) such that there is a path linking v_i to w_i and one linking v_i to w_{i-1} for every $i = 0, \dots, k-1$ (the indices being taken modulo k), and \vec{C} is the union of these paths. We use the notation $\vec{C} = (v_0 \Rightarrow$

$w_0 \leftarrow \cdots v_{k-1} \Rightarrow w_{k-1} \leftarrow v_0$). This is illustrated on Figure 3.1 where the set of sources on \vec{C} is $A = \{h, d\}$ and the set of sinks is $B = \{c, f\}$. Notice that the notation is not unique since $(h \Rightarrow f \Leftarrow d \Rightarrow c \Leftarrow h) = (d \Rightarrow c \Leftarrow h \Rightarrow f \Leftarrow d) = (h \Rightarrow c \Leftarrow d \Rightarrow f \Leftarrow h) = (d \Rightarrow f \Leftarrow h \Rightarrow c \Leftarrow d)$ in the example of Figure 3.1. Notice also that if \vec{C} is a dicycle in a digraph \vec{G} , then a source (sink) in \vec{C} is not necessarily a source in \vec{G} , as illustrated by vertex d in Figure 3.1.

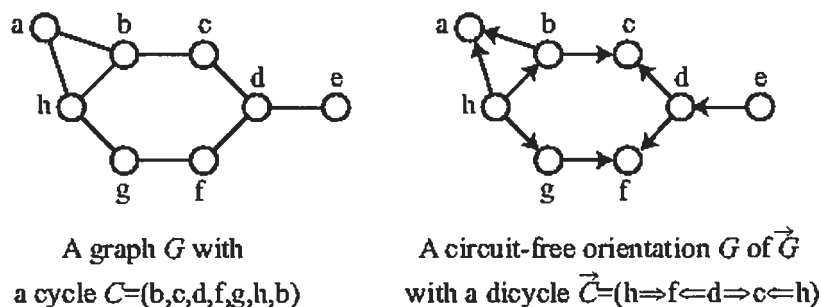


Figure 3.1: Illustration of a notation

Finally, we denote $\Omega(G)$ the set of circuit-free orientations of G and, for a fixed integer k , we denote $\Pi_k(G)$ the set of circuit-free orientations \vec{H} of a partial graph H of G such that $\lambda(\vec{H}) \leq k$.

Gallai [Gal68], Roy [Roy67] and Vitaver [Vit62] have independently proven the following famous theorem.

Theorem 6. [Gal68, Roy67, Vit62] *The length of a longest path in an orientation of a graph G is at least equal to the chromatic number of G .*

It follows from Theorem 6 that the problem of determining the chromatic number of a graph G is equivalent to the problem of orienting the edges of G so that the resulting digraph \vec{G} is circuit-free and $\lambda(\vec{G})$ is minimum. Indeed, given a $\chi(G)$ -coloring c of a graph G , one can easily construct a circuit-free orientation \vec{G}

with $\lambda(\vec{G}) = \chi(G)$ by simply orienting each edge (u, v) from u to v if and only if $c(u) < c(v)$. Conversely, given a circuit-free orientation \vec{G} of G , one can build a $\lambda(\vec{G})$ -coloring of G by assigning to each vertex v a color $c(v)$ equal to the length $d_{\vec{G}}^-(v)$ of a longest path ending at v in \vec{G} . Hence, one can state the following corollary of Theorem 6.

Corollary 7.
$$\chi(G) = \min_{\vec{G} \in \Omega(G)} \lambda(\vec{G})$$

It is interesting to note that the problem of determining a graph in $\Omega(G)$ with maximum $\lambda(\vec{G})$ is also NP-hard since any algorithm that would solve this problem would also solve the Hamiltonian path problem which is known to be NP-hard [GJ79]. Indeed, a graph G contains a Hamiltonian path if and only if its edges can be oriented so that the resulting digraph \vec{G} is circuit-free and $\lambda(\vec{G}) = |V(G)|$.

Since exact methods can only solve small graph coloring instances, heuristic methods are needed to get an upper bound on the chromatic number. As shown in a recent survey [GH06], most recent efficient graph coloring heuristics are either local search methods (e.g., tabu search, simulated annealing) or hybrid algorithms that combine a local search with a population-based method (e.g., genetic algorithm, adaptive memory programming). Barbosa et al. [BAdN04] have implemented a genetic algorithm that aims to minimize $\lambda(\vec{G})$. They consider a population of graph orientations chosen in $\Omega(G)$. Each member of the population is coded as a partial order of the vertices, where $u \rightarrow v$ means that u precedes v . A crossover operator combines two partial orders of the vertices for creating new members in the population (i.e, new graph orientations $\vec{G} \in \Omega(G)$). To our knowledge, no other evolutionary graph coloring algorithm based on edge orienting methods has ever been proposed, and we are not aware of any local search method that uses graph orientations to determine the chromatic number of a graph. The aim of this paper is to help bridging this gap by providing an analysis of the main ingredients that may be useful in the design of a local search graph coloring algorithm based on

edge orienting methods.

Local search methods can be described as follows. Let S be the set of solutions to a combinatorial optimization problem. For a solution $s \in S$, let $N(s)$ denote the *neighborhood* of s which is defined as the set of *neighbor solutions* in S obtained from s by performing a *local change* on it. Local search techniques visit a sequence s_0, \dots, s_n of solutions, where s_0 is an initial solution and $s_{i+1} \in N(s_i)$ ($i = 1, \dots, n-1$). When designing a local search algorithm for solving a particular problem, one has to define the search space to be explored, the evaluation function to be minimized, and the neighborhood function. This triplet is called a *search strategy*. In their recent survey, Galinier and Hertz [GH06] propose to classify the search strategies that have proven to be efficient for the graph coloring problem into four categories :

- The *legal strategy*: The search space S contains all legal colorings and the goal is to find a solution $c \in S$ that uses as few colors as possible.
- The *k-fixed partial legal strategy*: The number k of colors is fixed, the search space S contains all partial legal k -colorings, and the goal is to determine a solution $c \in S$ in which all vertices are colored.
- The *k-fixed penalty strategy*: The number k of colors is fixed, the search space S contains all (not necessarily legal) k -colorings, and the goal is to determine a legal k -coloring $c \in S$.
- The *penalty strategy*: The search space S contains all (not necessarily legal) colorings and the goal is to determine a legal coloring $c \in S$ that uses as few colors as possible.

There are at least two additional categories of local search strategies based on edge orienting methods that have never been tested and could prove successful for graph coloring.

- The *total orienting strategy*: The search space S contains all circuit-free graph orientations \vec{G} of G (i.e. $S = \Omega(G)$), and the goal is to minimize $\lambda(\vec{G})$.

- The *k-fixed partial orienting strategy*: An upper bound k on $\lambda(\vec{G})$ is fixed, the search space S contains all circuit-free graph orientations \vec{H} of a partial graph H of G with $\lambda(\vec{H}) \leq k$ (i.e., $S = \Pi_k(G)$), and the aim is to determine a solution \vec{H} having a maximum number of arcs.

For the *total orienting strategy*, a neighbor of a solution $\vec{H} \in \Omega(G)$ can be obtained by modifying the orientation of a subset of arcs, without creating circuits. For the *k-fixed partial orienting strategy*, a neighbor of a solution $\vec{H} \in \Pi_k(G)$ can be obtained by adding and orienting some edges that belong to G but not to H , and by then removing some arcs (if necessary) so that the resulting digraph also belongs to $\Pi_k(G)$. In summary, neighbor solutions in the above orienting strategies are obtained by changing the orientation of some arcs, or by adding some arcs and removing some others. We study in the next section some properties related to such changes. In particular, we characterize those that produce a circuit-free neighbor \vec{H} , and we compare $\lambda(\vec{G})$ with $\lambda(\vec{H})$. This theoretical study will help us to design and implement four basic tabu search algorithms for graph coloring based on orienting strategies. The four algorithms are described in Section 3.5 and compared numerically in Section 3.6 in order to determine which ones are the most promising. The algorithms we present illustrate the kind of methods one can design based on the theoretical results presented in Section 3.4. The development of more elaborate approaches is a topic for future study.

3.4 Properties

We start this section by giving some properties related to the modification of the orientation of some arcs in a digraph.

Property 8. *Let \vec{G} be a circuit-free orientation of a graph G and let \vec{H} be the digraph obtained from \vec{G} by changing the orientation of a subset F of arcs. Then \vec{H} contains a circuit if and only if there is a dicycle $\vec{C} = (v_0 \Rightarrow w_0 \Leftarrow \dots v_{k-1} \Rightarrow w_{k-1} \Leftarrow v_0)$ in \vec{G} such that no arc on the paths from v_i to w_i belongs to F and*

every arc on the paths from v_i to w_{i-1} belongs to F ($i = 0, \dots, k-1$) (indices being taken modulo k).

Proof. Assume there is a dicycle $\vec{C} = (v_0 \Rightarrow w_0 \Leftarrow \dots v_{k-1} \Rightarrow w_{k-1} \Leftarrow v_0)$ in \vec{G} such that no arc on the paths from v_i to w_i belongs to F and every arc on the paths from v_i to w_{i-1} belongs to F . Then \vec{C} is transformed into a circuit in \vec{H} .

Suppose now that \vec{H} contains a circuit $(x_0 \rightarrow \dots x_{r-1} \rightarrow x_0)$. Let I be the set containing all vertices x_i on the circuit such that $x_i \rightarrow x_{i-1} \in F$ and $x_{i+1} \rightarrow x_i \notin F$. Similarly, define J as the set containing all x_j with $x_j \rightarrow x_{j-1} \notin F$ and $x_{j+1} \rightarrow x_j \in F$. Then the circuit in \vec{H} corresponds to a dicycle \vec{C} in \vec{G} with I as set of sources and J as set of sinks. Hence, $\vec{C} = (v_0 \Rightarrow w_0 \Leftarrow \dots v_{k-1} \Rightarrow w_{k-1} \Leftarrow v_0)$ with $I = \{v_0, \dots, v_{k-1}\}$ and $J = \{w_0, \dots, w_{k-1}\}$. \square

The next two corollaries are about the special case where $|F| = 1$.

Corollary 9. *Let \vec{G} be a circuit-free orientation of a graph G and let \vec{H} be the digraph obtained from \vec{G} by changing the orientation of exactly one arc from $u \rightarrow v$ to $v \rightarrow u$. Then \vec{H} contains a circuit if and only there is a path of length at least 3 linking u to v in \vec{G} .*

Proof. According to Property 8, \vec{H} contains a circuit if and only if there is a dicycle $(v_0 \Rightarrow w_0 \Leftarrow v_0)$ in \vec{G} . Since G has no multiple edges, the path from v_0 to w_0 in \vec{G} has length at least 3. \square

Corollary 10. *If \vec{H} is obtained from a circuit-free orientation \vec{G} of G by changing the orientation of exactly one arc $u \rightarrow v$ with $d_{\vec{G}}^-(v) = d_{\vec{G}}^-(u) + 1$, then \vec{H} is circuit-free.*

Proof. From the previous Corollary, it is sufficient to observe that there is no path of length > 2 linking u to v in \vec{G} , else $d_{\vec{G}}^-(v) \geq d_{\vec{G}}^-(u) + 2$. \square

The next properties deal with the special case where F contains arcs lying on longest paths. The first part of Property 11 (i.e., \vec{H} is circuit-free) was also proved in [LAL92].

Property 11. *If \vec{H} is obtained from a circuit-free orientation \vec{G} of G by changing the orientation of exactly one arc $u \rightarrow v$ on a longest path, then*

- \vec{H} is circuit-free
- $\lambda(\vec{H}) \leq \lambda(\vec{G}) + 2$.
- if $\lambda(\vec{H}) > \lambda(\vec{G})$ then at least one of the two following situations must occur:
 - there is a second arc entering v that also belongs to a longest path in \vec{G} ;
 - there is a second arc leaving u that also belongs to a longest path in \vec{G} .

Proof. Since $u \rightarrow v$ is on a longest path in \vec{G} , there is no path of length > 2 linking u to v , and we therefore know from Corollary 9 that \vec{H} is circuit-free.

Since $d_{\vec{G}}^+(u) = d_{\vec{G}}^+(v) + 1$, $d_{\vec{G}}^-(v) = d_{\vec{G}}^-(u) + 1$, $d_{\vec{H}}^-(v) \leq d_{\vec{G}}^-(v)$, and $d_{\vec{H}}^+(u) \leq d_{\vec{G}}^+(u)$, we know that the length of a longest path in \vec{H} that contains $v \rightarrow u$ has length $d_{\vec{H}}^-(v) + d_{\vec{H}}^+(u) \leq d_{\vec{G}}^-(u) + d_{\vec{G}}^+(v) + 2 = \lambda(\vec{G}) + 2$. Since all paths in \vec{H} that do not contain $v \rightarrow u$ have a length $\leq \lambda(\vec{G})$, we conclude that $\lambda(\vec{H}) \leq \lambda(\vec{G}) + 2$.

If $\lambda(\vec{H}) > \lambda(\vec{G})$, then every longest path in \vec{H} necessarily contains the arc $v \rightarrow u$, hence $\lambda(\vec{H}) = d_{\vec{H}}^-(v) + d_{\vec{H}}^+(u)$. If $u \rightarrow v$ is the unique arc entering v and the unique arc leaving u that belongs to a longest path in \vec{G} , then $d_{\vec{H}}^-(v) \leq d_{\vec{G}}^-(v) - 1$ and $d_{\vec{H}}^+(u) \leq d_{\vec{G}}^+(u) - 1$, which means that $\lambda(\vec{H}) \leq d_{\vec{G}}^-(v) + d_{\vec{G}}^+(u) - 2 = \lambda(\vec{G})$, a contradiction. \square

When designing a local search algorithm, it is important to ensure that given any initial solution s_0 , there is a sequence of neighbor solutions leading to an optimal one (i.e., a sequence s_0, \dots, s_n with $s_i \in N(s_{i-1})$ and where s_n is optimal).

The next property demonstrates that this requirement is satisfied for the *total orienting strategy* when a neighbor is obtained by modifying the orientation of one arc on a longest path. For a digraph $\vec{H} \in \Omega(G)$, let $N_1(\vec{H})$ denote the set of digraphs that can be obtained from \vec{G} by modifying the orientation of exactly one arc on a longest path in \vec{H} . We know from Property 11 that every digraph in $N_1(\vec{H})$ belongs to $\Omega(G)$.

Property 12. *For every digraph $\vec{G} \in \Omega(G)$ there is a sequence $(\vec{H}_0, \dots, \vec{H}_n)$ of digraphs such that $\vec{H}_0 = \vec{G}$, $\lambda(\vec{H}_n) = \chi(G)$, and each \vec{H}_i belongs to $N_1(\vec{H}_{i-1})$.*

Proof. Let \vec{G}^* be an orientation of G with $\lambda(\vec{G}^*) = \chi(G)$. Also, for an orientation $\vec{H}_i \in \Omega(G)$, let $Q(\vec{H}_i)$ denote the set of arcs in \vec{H}_i that have the opposite orientation in \vec{G}^* . If $\lambda(\vec{H}_i) > \chi(G)$, then every longest path in \vec{H}_i contains at least one arc in $Q(\vec{H}_i)$. Let \vec{H}_{i+1} be the digraph obtained from \vec{H}_i by changing the orientation of one such arc. One can repeat this process until one reaches a digraph \vec{H}_n with $\lambda(\vec{H}_n) = \chi(G)$. This will occur in at most $|Q(\vec{H}_0)|$ steps since $|Q(\vec{H}_{i+1})| = |Q(\vec{H}_i)| - 1$ and $\vec{H}_n = \vec{G}^*$ when $|Q(\vec{H}_n)| = 0$. \square

Notice that for reaching an optimal solution \vec{H}_n starting from a solution \vec{H}_0 , it can be necessary to visit a solution \vec{H}_i with $\lambda(\vec{H}_i) > \lambda(\vec{H}_0)$. This is illustrated in Figure 3.2. The digraph \vec{H}_0 has two longest paths $(e \rightarrow d \rightarrow b \rightarrow a)$ and $(e \rightarrow c \rightarrow b \rightarrow a)$ of length 4. One can easily verify that by changing the orientation of $e \rightarrow d$, $e \rightarrow c$, $d \rightarrow b$, or $c \rightarrow b$, one gets a digraph \vec{H} with a longest path $(c \rightarrow e \rightarrow d \rightarrow b \rightarrow a)$ of length $\lambda(\vec{H}) = 5$. The unique arc on a longest path in \vec{H}_0 that can be reversed without increasing the length of a longest path is the arc $b \rightarrow a$. But the digraph obtained by reversing this arc is similar to \vec{H}_0 , where a and b have exchanged their role. Hence, in order to reach an optimal orientation \vec{H}_n of G with $\lambda(\vec{H}_n) = 3$ (as the rightmost digraph of Figure 3.2 that contains two longest paths $(c \rightarrow b \rightarrow a)$ and $(d \rightarrow b \rightarrow a)$), one has to visit a digraph with longest path of length $5 > 4 = \lambda(\vec{H}_0)$.

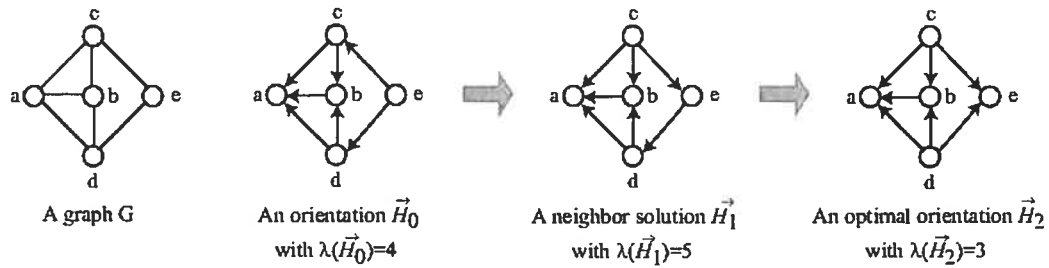


Figure 3.2: A sequence of neighbor solutions

For a digraph \vec{G} , let $W(\vec{G})$ denote its set of vertices on a longest path. Also, for a vertex $v \in W(\vec{G})$, let $A_{\vec{G}}^-(v)$ and $A_{\vec{G}}^+(v)$ denote the set of arcs on longest paths in \vec{G} having v as head and as tail, respectively.

Property 13. *Let \vec{G} be a circuit-free orientation of G , and let v be a vertex in $W(\vec{G})$. If \vec{H} is obtained from \vec{G} by changing the orientation of a subset of arcs in $A_{\vec{G}}^-(v)$ or a subset of arcs in $A_{\vec{G}}^+(v)$, then \vec{H} is also circuit-free.*

Proof. Let F be the set of arcs in \vec{G} that have been modified in \vec{G} to get \vec{H} , and assume, by contradiction, that \vec{H} contains a circuit. We know from Property 8 that there is a dicycle $\vec{C} = (v_0 \Rightarrow w_0 \Leftarrow \cdots v_{k-1} \Rightarrow w_{k-1} \Leftarrow v_0)$ in \vec{G} such that no arc on the paths from v_i to w_i belongs to F and every arc on the paths from v_i to w_{i-1} belongs to F . Since all arcs in F have the same tail (or the same head), such a dicycle \vec{C} contains exactly one arc of F , which means that \vec{C} is the union of the arc $v_0 \rightarrow w_0$ in F with a directed path \vec{P} linking v_0 to w_0 having no arc in F . Since G has no multiple edges, \vec{P} has length at least three, which contradicts the fact that $v_0 \rightarrow w_0$ is on a longest path in \vec{G} . \square

Property 14. *Let \vec{G} be a circuit-free orientation of G , and let v be a vertex in $W(\vec{G})$. If \vec{H} is obtained from \vec{G} by changing the orientation of all arcs in $A_{\vec{G}}^-(v)$ or all arcs in $A_{\vec{G}}^+(v)$, then*

- \vec{H} is circuit-free;
- $\lambda(\vec{H}) \leq \lambda(\vec{G}) + 1$.

Proof. We know from Property 13 that \vec{H} is circuit-free. Let \vec{P} be a longest path in \vec{H} . If all arcs on \vec{P} have the same orientation as in \vec{G} , then $\lambda(\vec{H}) \leq \lambda(\vec{G})$. Otherwise, let F be the set of arcs that have been modified in \vec{G} to get \vec{H} . Since all arcs in F have the same tail or the same head, \vec{P} contains exactly one arc $x \rightarrow y$ that was originally oriented from y to x in \vec{G} . Hence, $v = x$ and $F = A_{\vec{G}}^-(v)$, or $v = y$ and $F = A_{\vec{G}}^+(v)$. We therefore have either $d_{\vec{H}}^-(x) < d_{\vec{G}}^-(x)$ and $d_{\vec{H}}^+(y) \leq d_{\vec{G}}^+(y)$, or $d_{\vec{H}}^+(y) < d_{\vec{G}}^+(y)$ and $d_{\vec{H}}^-(x) \leq d_{\vec{G}}^-(x)$. So $\lambda(\vec{H}) = d_{\vec{H}}^-(x) + d_{\vec{H}}^+(y) < d_{\vec{G}}^-(x) + d_{\vec{G}}^+(y) = \lambda(\vec{G}) + 2$. \square

We now prove a result similar to Property 12. For a digraph $\vec{H} \in \Omega(G)$, let $N_2(\vec{H})$ denote the set of digraphs that can be obtained from \vec{G} by changing the orientation of all arcs in $A_{\vec{G}}^-(v)$ or all arcs in $A_{\vec{G}}^+(v)$, for some $v \in W(\vec{G})$. We know from Property 14 that every digraph in $N_2(\vec{H})$ belongs to $\Omega(G)$.

Property 15. *For every $\vec{G} \in \Omega(G)$ there is a sequence $(\vec{H}_0, \dots, \vec{H}_n)$ of digraphs such that $\vec{H}_0 = \vec{G}$, $\lambda(\vec{H}_n) = \chi(G)$, and each \vec{H}_i belongs to $N_2(\vec{H}_{i-1})$.*

Proof. Let \vec{G}^* be an orientation of G with $\lambda(\vec{G}^*) = \chi(G)$. Also, for an orientation $\vec{H}_i \in \Omega(G)$, let $Q(\vec{H}_i)$ denote the set of arcs in \vec{H}_i that have the opposite orientation in \vec{G}^* . If $\lambda(\vec{H}_i) > \chi(G)$, then there is a vertex $v \in W(\vec{H}_i)$ such that all arcs in $A_{\vec{H}_i}^+(v)$ have the opposite orientation in \vec{G}^* . Indeed, assume by contradiction that each vertex $v \in W(\vec{H}_i)$ with $A_{\vec{H}_i}^+(v) \neq \emptyset$ is the tail of at least one arc $v \rightarrow w$ in \vec{H}_i that has the same orientation as in \vec{G}^* . Consider any vertex v_1 in \vec{H}_i with $d_{\vec{H}_i}^+(v_1) = \lambda(\vec{H}_i)$. Then each vertex v_j ($j = 1, \dots, \lambda(\vec{H}_i) - 1$) is the tail of an arc $v_j \rightarrow v_{j+1}$ in $A_{\vec{H}_i}^+(v_j)$ that has the same orientation as in \vec{G}^* . Hence $(v_1, \dots, v_{\lambda(\vec{H}_i)})$ is a path of length $\lambda(\vec{H}_i) > \lambda(\vec{G}^*)$ in \vec{G}^* , a contradiction.

So let v be a vertex in $W(\vec{H}_i)$ such that all arcs in $A_{\vec{H}_i}^+(v)$ have the opposite orientation in \vec{G}^* , and let \vec{H}_{i+1} be the digraph obtained from \vec{H}_i by changing the orientation of all arcs in $A_{\vec{H}_i}^+(v)$. One can repeat this process until one reaches a digraph \vec{H}_n with $\lambda(\vec{H}_n) = \chi(G)$. This will occur in at most $|Q(\vec{H}_0)|$ steps since $|Q(\vec{H}_{i+1})| < |Q(\vec{H}_i)|$ and $\vec{H}_n = \vec{G}^*$ when $|Q(\vec{H}_n)| = 0$. \square

Notice that we could have proven the above Property by showing, in a very similar way, that if $\lambda(\vec{H}_i) > \chi(G)$, then there is a vertex $v \in W(\vec{H}_i)$ such that all arcs in $A_{\vec{H}_i}^-(v)$ have the opposite orientation in \vec{G}^* .

For a digraph \vec{G} and an integer $\ell \in \{1, \dots, \lambda(\vec{G}) - 1\}$, let $\vec{L}_\ell(\vec{G})$ denote the partial digraph of \vec{G} containing only those arcs $u \rightarrow v$ on longest paths such that $d_{\vec{G}}^-(u) = \ell$ and $d_{\vec{G}}^-(v) = \ell + 1$.

Property 16. *Let \vec{G} be a circuit-free orientation of a graph G , and consider any integer $\ell \in \{1, \dots, \lambda(\vec{G}) - 1\}$. If \vec{H} is obtained from \vec{G} by changing the orientation of all arcs in a subset of connected components of $\vec{L}_\ell(\vec{G})$, then*

- \vec{H} is circuit-free;
- $\lambda(\vec{H}) \leq \lambda(\vec{G})$.

Proof. Let F denote the set of arcs that have been modified in \vec{G} to obtain \vec{H} . If \vec{H} contains a circuit, then we know from Property 8 that there is a dicycle $\vec{C} = (v_0 \Rightarrow w_0 \Leftarrow \dots \Leftarrow v_{k-1} \Rightarrow w_{k-1} \Leftarrow v_0)$ in \vec{G} such that no arc on the paths from v_i to w_i belongs to F and every arc on the paths from v_i to w_{i-1} belongs to F . Hence, $d_{\vec{G}}^-(v_i) = \ell$ and $d_{\vec{G}}^-(w_i) = \ell + 1$ for $i = 0, \dots, k - 1$, which means that each path from v_i to w_i and from v_i to w_{i-1} contains a single arc. Since G has no multiple edges, we have $k > 0$. So there are two longest paths $(x_1 \rightarrow \dots \rightarrow x_\ell = v_0 \rightarrow x_{\ell+1} = w_{k-1} \rightarrow \dots \rightarrow x_{\lambda(\vec{G})})$ and $(y_1 \rightarrow \dots \rightarrow y_\ell = v_1 \rightarrow y_{\ell+1} = w_0 \rightarrow \dots \rightarrow y_{\lambda(\vec{G})})$ in \vec{G} containing $v_0 \rightarrow w_{k-1}$ and $v_1 \rightarrow w_0$, respectively. But this means that $(x_1 \rightarrow \dots \rightarrow x_\ell = v_0 \rightarrow y_{\ell+1} = w_0 \rightarrow \dots \rightarrow y_{\lambda(\vec{G})})$ is a longest path in \vec{G} containing

$v_0 \rightarrow w_0$, which implies that $v_0 \rightarrow w_0$ is in the same connected component of $\vec{L}_\ell(\vec{G})$ as $v_0 \rightarrow w_{k-1}$ and $v_1 \rightarrow w_0$. Hence, $v_0 \rightarrow w_0 \in F$, a contradiction. So \vec{H} is circuit-free.

Assume now $\lambda(\vec{H}) > \lambda(\vec{G})$ and let $\vec{P} = (x_1 \rightarrow \cdots \rightarrow x_k)$ be a path in \vec{H} of length $k > \lambda(\vec{G})$. Let I denote the set of indices i such that $x_{i+1} \rightarrow x_i \in F$. We have $|I| \geq 1$, else \vec{P} also exists in \vec{G} . So let i be the smallest element in I . This means that $d_{\vec{G}}^-(x_i) = \ell + 1$, $d_{\vec{G}}^-(x_{i+1}) = \ell$, and there is a longest path $(y_1 \rightarrow \cdots \rightarrow y_\ell = x_{i+1} \rightarrow y_{\ell+1} = x_i \rightarrow \cdots \rightarrow y_{\lambda(\vec{G})})$ in \vec{G} containing the arc $x_{i+1} \rightarrow x_i$. Suppose $|I| > 1$ and let j be the smallest element in I larger than i . Then there is a longest path $(z_1 \rightarrow \cdots \rightarrow z_\ell = x_{j+1} \rightarrow z_{\ell+1} = x_j \rightarrow \cdots \rightarrow z_{\lambda(\vec{G})})$ in \vec{G} containing the arc $x_{j+1} \rightarrow x_j$. Since all arcs $x_r \rightarrow x_{r+1}$ of \vec{P} with $i < r < j$ are also in \vec{G} , while $d_{\vec{G}}^-(x_{i+1}) = \ell$ and $d_{\vec{G}}^-(x_j) = \ell + 1$, we necessarily have $j = i + 2$. Hence, $(y_1 \rightarrow \cdots \rightarrow y_\ell = x_{i+1} \rightarrow z_{\ell+1} = x_{i+2} \rightarrow \cdots \rightarrow z_{\lambda(\vec{G})})$ is a longest path in \vec{G} containing the arc $x_{i+1} \rightarrow x_{i+2}$. But this means that $x_{i+1} \rightarrow x_{i+2}$ is in the same connected component of $\vec{L}_\ell(\vec{G})$ as $x_{i+1} \rightarrow x_i$ and $x_{j+1} \rightarrow x_j$. Hence, $x_{i+1} \rightarrow x_{i+2} \in F$, a contradiction.

So we know that $I = \{i\}$ (i.e., I contains exactly one element). Since the path $(x_1 \rightarrow \cdots \rightarrow x_i = y_{\ell+1} \rightarrow \cdots \rightarrow y_{\lambda(\vec{G})})$ in \vec{G} is of length $i + \lambda(\vec{G}) - (\ell + 1)$, we necessarily have $i \leq \ell + 1$. If $i = \ell + 1$ then $x_{i-1} \rightarrow x_i$ is in the same connected component of $\vec{L}_\ell(\vec{G})$ as $x_{i+1} \rightarrow x_i$, which means that $x_{i-1} \rightarrow x_i \in F$, a contradiction. We therefore have $i \leq \ell$. Moreover, since $(y_1 \rightarrow \cdots \rightarrow y_\ell = x_{i+1} \rightarrow \cdots \rightarrow x_k)$ is a path of length $\ell + k - (i + 1)$ in \vec{G} , we have $k - i \leq \lambda(\vec{G}) - \ell + 1$. Also, if $k - i = \lambda(\vec{G}) - \ell + 1$ then $x_{i+1} \rightarrow x_{i+2}$ is in the same connected component of $\vec{L}_\ell(\vec{G})$ as $x_{i+1} \rightarrow x_i$, which means that $x_{i+1} \rightarrow x_{i+2} \in F$, a contradiction. So finally, $k - i \leq \lambda(\vec{G}) - \ell$, which implies that \vec{P} is a path of length $k = i + (k - i) \leq \ell + \lambda(\vec{G}) - \ell = \lambda(\vec{G})$, a contradiction. \square

Let $N_3(\vec{H})$ denote the set of digraphs that can be obtained from \vec{H} by choosing an integer $\ell \in \{1, \dots, \lambda(\vec{H}) - 1\}$, and by changing the orientation of all arcs in

a subset of connected components of $\vec{L}_\ell(\vec{H})$. One could think about proving a result similar to Properties 12 and 15, but with $\vec{H}_i \in N_3(\vec{H}_{i-1})$. However, the example in Figure 3.3 demonstrates that there are digraphs $\vec{H} \in \Omega(G)$ such that there are no sequences $(\vec{H}_0, \dots, \vec{H}_n)$ with $\vec{H}_0 = \vec{H}$, $\vec{H}_i \in N_3(\vec{H}_{i-1})$ ($i = 1, \dots, n$), and $\lambda(\vec{H}_n) = \chi(G)$. The left graph corresponds to a digraph \vec{H} with $\lambda(\vec{H}) = 4$. In order not to overload the drawing, we indicate the value $d_{\vec{H}}^-(v)$ of each vertex v instead of representing each arc as a directed line. In other words, each arc should be directed from the smallest label to the largest one. As shown in Figure 3.3, each digraph $\vec{L}_\ell(\vec{H})$ ($\ell = 1, 2, 3$) contains exactly one connected component, and it is easy to verify that all digraphs in $N_3(\vec{G})$ are equivalent to \vec{H} , where the roles of the vertices with label $d_{\vec{H}}^-(v) = \ell$ are permuted with those with label $d_{\vec{H}}^-(v) = \ell + 1$. In summary, all digraphs in a sequence $(\vec{H}_0, \vec{H}_1, \dots)$ with $\vec{H}_0 = \vec{H}$ and $\vec{H}_i \in N_3(\vec{H}_{i-1})$ are equivalent to \vec{H} and have a longest path of length $\lambda(\vec{H}_i) = 4$. However, an optimal orientation \vec{G} of G with $\lambda(\vec{G}) = 3$ is represented on the right of Figure 3.3.

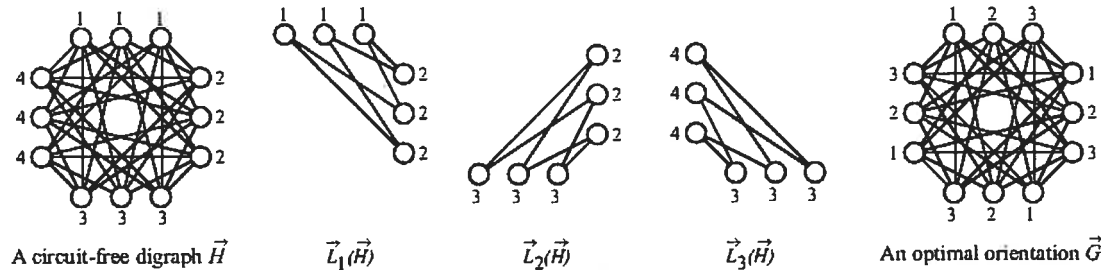


Figure 3.3: Illustration of the *total orienting strategy* with neighborhood N_3

We now give properties related to the addition or the removal of arcs.

Property 17. *Let \vec{G} be a circuit-free orientation of a graph G , and let u and v be two non-adjacent vertices in G . If $d_{\vec{G}}^-(u) \leq d_{\vec{G}}^-(v) + 1$ then the graph \vec{H} obtained by adding an arc $u \rightarrow v$ in \vec{G} is circuit-free.*

Proof. If \vec{H} contains a circuit $\vec{C} = (x_0 \rightarrow \cdots \rightarrow x_k \rightarrow x_0)$, then \vec{C} contains the arc $u \rightarrow v$. Without loss of generality, we may assume $x_k = u$ and $x_0 = v$. But this means that $(x_0 \rightarrow \cdots \rightarrow x_k)$ is a path in \vec{G} . Since u is not adjacent to v in G , we have $k > 1$ which means that $d_{\vec{G}}^-(u) > d_{\vec{G}}^-(v) + 1$. \square

By adding an arc between two non-adjacent vertices in a digraph \vec{G} , one gets a digraph \vec{H} . If \vec{H} is circuit-free, then $\lambda(\vec{H}) \geq \lambda(\vec{G})$. The next property indicates when $\lambda(\vec{H})$ is strictly larger than $\lambda(\vec{G})$.

Property 18. *Let \vec{G} be a circuit-free orientation of a graph G , and let \vec{H} be a circuit-free digraph obtained from \vec{G} by adding an arc $u \rightarrow v$ between two non-adjacent vertices u and v of G . Then $\lambda(\vec{H}) > \lambda(\vec{G})$ if and only if $d_{\vec{G}}^-(u) + d_{\vec{G}}^+(v) > \lambda(\vec{G})$.*

Proof. $\lambda(\vec{H}) > \lambda(\vec{G})$ if and only if there is a longest path in \vec{H} containing $u \rightarrow v$ and having a length strictly larger than $\lambda(\vec{G})$. It is now sufficient to observe that the length of such a path is equal to $d_{\vec{G}}^-(u) + d_{\vec{G}}^+(v)$. \square

Corollary 19. *Let \vec{G} be a circuit-free orientation of a graph G , and let \vec{H} be a circuit-free digraph obtained from \vec{G} by adding an arc $u \rightarrow v$ between two non-adjacent vertices u and v of G with $d_{\vec{G}}^-(u) < d_{\vec{G}}^-(v)$. Then $\lambda(\vec{H}) = \lambda(\vec{G})$.*

Proof. Since $\lambda(\vec{G}) \geq d_{\vec{G}}^-(v) + d_{\vec{G}}^+(v) - 1$, we have $d_{\vec{G}}^-(u) + d_{\vec{G}}^+(v) \leq d_{\vec{G}}^-(u) - d_{\vec{G}}^-(v) + \lambda(\vec{G}) + 1 \leq \lambda(\vec{G})$. From Property 18, we conclude that $\lambda(\vec{H}) = \lambda(\vec{G})$. \square

A digraph \vec{H} in $\Pi_k(G)$ is said maximal if no arc can be added to \vec{H} so that the resulting digraph also belongs to $\Pi_k(G)$.

Property 20. Let \vec{H} be a maximal digraph in $\Pi_k(G)$, and let u and v be two vertices that are adjacent in G but not in H . Then $d_{\vec{H}}^-(u) = d_{\vec{H}}^-(v)$.

Proof. If $d_{\vec{H}}^-(u) < d_{\vec{H}}^-(v)$, then we know from Property 17 and Corollary 19 that the graph obtained by adding the arc $u \rightarrow v$ in \vec{H} also belongs to $\Pi_k(G)$. This contradicts the maximality of \vec{H} in $\Pi_k(G)$, and we therefore have $d_{\vec{H}}^-(u) \geq d_{\vec{H}}^-(v)$. By permuting the roles of u and v , we also have $d_{\vec{H}}^-(u) \leq d_{\vec{H}}^-(v)$. \square

Property 21. Let \vec{H} be a maximal digraph in $\Pi_k(G)$, and let \vec{H}' be a digraph obtained from \vec{H} by adding and orienting an edge that belongs to G but not to H . Then

- \vec{H}' is circuit-free;
- $\lambda(\vec{H}') > k$.

Proof. Let $u \rightarrow v$ be the arc that has been added to \vec{H} to obtain \vec{H}' . We know from Property 20 that $d_{\vec{H}}^-(u) = d_{\vec{H}}^-(v)$. Hence, there is no path from v to u in \vec{H} , which means that \vec{H}' is circuit-free. Since \vec{H} is maximal in $\Pi_k(G)$, we therefore have $\lambda(\vec{H}') > k$. \square

By removing some arcs from a digraph \vec{G} , one gets a digraph \vec{H} with $\lambda(\vec{H}) \leq \lambda(\vec{G})$. The next Property indicates when $\lambda(\vec{H})$ is strictly smaller than $\lambda(\vec{G})$.

Property 22. Let \vec{H} be the digraph obtained by removing a subset F of arcs from an orientation \vec{G} of G . Then $\lambda(\vec{H}) < \lambda(\vec{G})$ if and only if every longest path in \vec{G} has at least one arc in F .

Proof. If there is a longest path in \vec{G} with no arc in F , then this path also exists in \vec{H} which means that $\lambda(\vec{H}) = \lambda(\vec{G})$. Otherwise, all paths in \vec{H} have a length strictly smaller than $\lambda(\vec{G})$. \square

We close this section with a property that demonstrates a result similar to Properties 12 and 15, but for the *k-fixed partial orienting strategy*, where a neighbor is obtained by adding an arc and by removing some others so that the resulting digraph belongs to $\Pi_k(G)$. More precisely, for a digraph $\vec{H} \in \Pi_k(G)$, let $N_4(\vec{H})$ be the set of digraphs \vec{H}' constructed from \vec{H} as follows.

1. A maximal digraph \vec{D} in $\Pi_k(G)$ is first obtained from \vec{H} by adding arcs that create no circuits and no paths of length $> k$. Notice that $\vec{D} = \vec{H}$ if \vec{H} is already maximal in $\Pi_k(G)$.
2. If $\vec{D} \in \Omega(G)$, then the neighbor $\vec{H}' \in N_4(\vec{H})$ of \vec{H} is defined equal to \vec{D} .

If $\vec{D} \notin \Omega(G)$, a digraph \vec{D}' is built from \vec{D} by adding and orienting an edge that belongs to G but not to D . We know from Property 21 that \vec{D}' is circuit-free but contains paths of length $> k$. The neighbor $\vec{H}' \in N_4(\vec{H})$ of \vec{H} is then obtained from \vec{D}' by removing a minimal (inclusion-wise) set of arcs so that no path in \vec{H}' has length $> k$.

Property 23. *Let G be a graph and let k be any integer larger than or equal to $\chi(G)$. For every $\vec{H} \in \Pi_k(G)$ there is a sequence $(\vec{H}_0, \dots, \vec{H}_n)$ of digraphs such that $\vec{H}_0 = \vec{H}$, $\vec{H}_n \in \Omega(G)$, and each \vec{H}_i belongs to $N_4(\vec{H}_{i-1})$.*

Proof. Let \vec{G}^* be a circuit-free orientation of G with $\lambda(\vec{G}^*) \leq k$. Also, for a digraph $\vec{H} \in \Pi_k(G)$, let $B(\vec{H})$ denote the set of arcs in \vec{H} that are oriented as in \vec{G}^* . Let \vec{D}_i be a maximal digraph in $\Pi_k(G)$ obtained from \vec{H}_i by adding arcs that create no circuits and no paths of length $> k$. We have $|B(\vec{D}_i)| \geq |B(\vec{H}_i)|$. If $\vec{D}_i \in \Omega(G)$ then define \vec{H}_{i+1} equal to \vec{D}_i , else let \vec{D}'_i denote the digraph obtained from \vec{D}_i by adding an arc $u \rightarrow v$ of \vec{G}^* between two non adjacent vertices u and v in \vec{D}_i . We know from Property 21 that \vec{D}'_i has no circuits but has paths of length $> k$. These too long paths do not exist in \vec{G}^* and can therefore be removed from \vec{D}'_i by deleting a minimal (inclusion-wise) set of arcs $x \rightarrow y$ so that \vec{G}^*

contains the opposite arc $y \rightarrow x$. After all these removals, one gets a digraph \vec{H}_{i+1} with $|B(\vec{H}_{i+1})| = |B(\vec{D}'_i)| = |B(\vec{D}_i)| + 1 \geq |B(\vec{H}_i)| + 1$. One can repeat this process until one reaches a digraph $\vec{H}_n \in \Omega(G)$. This will occur in at most $|E(G)| - |B(\vec{H}_0)|$ steps since $\vec{H}_n = \vec{G}^*$ when $|B(\vec{H}_n)| = |E(G)|$. \square

3.5 Four tabu search algorithms

Tabu search is one of the most famous local search techniques. It was introduced by Glover in 1986. A description of the method and its concepts can be found in [GL97]. For a solution s in the search space S , its neighborhood $N(s)$ is defined as the set of solutions $s' \in S$ obtained from s by performing a *local change* m on it. We denote $s' = s \oplus m$. A basic tabu search is described in Figure 3.4.

Choose an initial solution s ; set $TL = \emptyset$ (tabu list); set $s^* = s$ (best solution)
Repeat the following until a stopping criterion is met

- Determine a best solution $s' \in N(s)$ such that either $s' = s \oplus m$ with $m \notin TL$ or s' is better than s^*
- If s' is better than s^* then set $s^* := s'$
- Set $s := s'$ and update TL

Figure 3.4: Basic tabu search

In this section, we describe four tabu search algorithms for the graph coloring problem. Three of them use the *total orienting strategy*, and one uses the *k-fixed partial orienting strategy*. In order to describe these algorithms, it is sufficient to clearly define the search space S , the objective function to be minimized, the neighborhood of each solution, and the contents and size of the tabu list TL .

For a circuit-free digraph \vec{G} , we denote

- $W(\vec{G})$ its set of vertices on longest paths;
- $L(\vec{G})$ its set of arcs on longest paths;

- $A_{\vec{G}}^-(v)$ the set of arcs in $L(\vec{G})$ having vertex v as head, and $A_{\vec{G}}^+(v)$ the set of arcs in $L(\vec{G})$ having vertex v as tail (see Section 3.4);
- $\vec{L}_\ell(\vec{G})$ the partial digraph of \vec{G} containing only those arcs $u \rightarrow v$ on longest paths such that $d_{\vec{G}}^-(u) = \ell$ and $d_{\vec{G}}^-(v) = \ell + 1$, where ℓ is an integer in $\{1, \dots, \lambda(\vec{G}) - 1\}$ (see Section 3.4);
- $N_1(\vec{G})$ the set of digraphs obtained from \vec{G} by changing the orientation of exactly one arc in $L(\vec{G})$;
- $N_2(\vec{G})$ the set of digraphs obtained from \vec{G} by choosing a vertex v in $W(\vec{G})$ and by changing the orientation of all arcs in $A_{\vec{G}}^-(v)$, or of all arcs in $A_{\vec{G}}^+(v)$;
- $N_3(\vec{G})$ the set of digraphs obtained from \vec{G} by choosing a number $\ell \in \{1, \dots, \lambda(\vec{G}) - 1\}$ and by changing the orientation of all arcs in a subset of connected components of $\vec{L}_\ell(\vec{G})$.

We know from Properties 11, 14, and 16 that all digraphs in $N_i(\vec{G})$ ($i = 1, 2, 3$) are circuit-free. Moreover,

- $\lambda(\vec{H}) \leq \lambda(\vec{G}) + 2$ for all $\vec{H} \in N_1(\vec{G})$;
- $\lambda(\vec{H}) \leq \lambda(\vec{G}) + 1$ for all $\vec{H} \in N_2(\vec{G})$;
- $\lambda(\vec{H}) \leq \lambda(\vec{G})$ for all $\vec{H} \in N_3(\vec{G})$.

The first tabu search algorithm, called TABU.1, has $\Omega(G)$ as search space and uses neighborhood N_1 . The objective is to determine an orientation \vec{G} of G with minimum $\lambda(\vec{G})$. Since many orientations of G may have longest paths of the same length, we discriminate two orientations \vec{G} and \vec{H} with $\lambda(\vec{G}) = \lambda(\vec{H})$ by counting their number of arcs on longest paths. More precisely, a solution \vec{G} is better than a solution \vec{H} if and only if $\lambda(\vec{G}) < \lambda(\vec{H})$ or $\lambda(\vec{G}) = \lambda(\vec{H})$ and $|L(\vec{G})| < |L(\vec{H})|$. The tabu list contains edges with the meaning that it is forbidden to change their orientation. When performing a move from a solution \vec{G} to a neighbor solution $\vec{H} \in N_1(\vec{G})$, the orientation of an arc $u \rightarrow v$ is changed and the edge (u, v) is introduced in the tabu list TL . We have proved in Property 12

that for every $\vec{G} \in \Omega(G)$ there is a sequence $(\vec{H}_0, \dots, \vec{H}_n)$ of digraphs such that $\vec{H}_0 = \vec{G}$, $\lambda(\vec{H}_n) = \chi(G)$, and each \vec{H}_i belongs to $N_1(\vec{H}_{i-1})$ ($i = 1, \dots, n$).

The second tabu search algorithm, called TABU_2, also has $\Omega(G)$ as search space but uses neighborhood N_2 . As for TABU_1, a solution \vec{G} is better than a solution \vec{H} if and only if $\lambda(\vec{G}) < \lambda(\vec{H})$ or $\lambda(\vec{G}) = \lambda(\vec{H})$ and $|L(\vec{G})| < |L(\vec{H})|$. When a neighbor $\vec{H} \in N_2(\vec{G})$ of a solution \vec{G} is obtained by changing the orientation of all arcs $u \rightarrow v$ in $A_{\vec{G}}^-(v)$, the pair $(v, +)$ is introduced in the tabu list TL , with the meaning that it is now forbidden for several iterations to change the orientation of an arc having v as tail. Similarly, when \vec{H} is obtained from \vec{G} by changing the orientation of all arcs $v \rightarrow w$ in $A_{\vec{G}}^+(v)$, the pair $(v, -)$ is introduced in TL in order to forbid the change of the orientation of an arc having v as head. We have proved in Property 15 that for every $\vec{G} \in \Omega(G)$ there is a sequence $(\vec{H}_0, \dots, \vec{H}_n)$ of digraphs such that $\vec{H}_0 = \vec{G}$, $\lambda(\vec{H}_n) = \chi(G)$, and each \vec{H}_i belongs to $N_2(\vec{H}_{i-1})$ ($i = 1, \dots, n$).

The third tabu search algorithm that uses the *total orienting strategy* is called TABU_3. As for TABU_1 and TABU_2, the solution space S is $\Omega(G)$, and a solution \vec{G} is better than a solution \vec{H} if and only if $\lambda(\vec{G}) < \lambda(\vec{H})$ or $\lambda(\vec{G}) = \lambda(\vec{H})$ and $|L(\vec{G})| < |L(\vec{H})|$. However, the neighborhood in this case is N_3 . The tabu list TL contains edges with the meaning that it is forbidden to move from a solution \vec{G} to a solution $\vec{H} \in N_3(\vec{G})$ if such a move requires to change the orientation of at least one edge in TL . When performing a move from a solution \vec{G} to a neighbor solution $\vec{H} \in N_3(\vec{G})$, some edges change their orientation, and all these edges are introduced in the tabu list TL . As shown in the previous section, there are orientations $\vec{G} \in \Omega(G)$ for which there is no sequence $(\vec{H}_0, \dots, \vec{H}_n)$ of digraphs with $\vec{H}_0 = \vec{G}$, $\lambda(\vec{H}_n) = \chi(G)$, and each \vec{H}_i belongs to $N_3(\vec{H}_{i-1})$.

The fourth and last tabu search algorithm, called TABU_4, is based on the *k-fixed partial orienting strategy*. It works with a fixed integer k and tries to determine an orientation \vec{G} of a given graph G so that $\lambda(\vec{G}) \leq k$. The search space S is $\Pi_k(G)$, and a solution is better than another one if and only if it contains more arcs. A neighbor $\vec{H}' \in N_4(\vec{H})$ of a solution \vec{H} is obtained as follows (see also Section 3.4).

A maximal digraph \vec{D} in $\Pi_k(G)$ is first obtained from \vec{H} by adding arcs that create no circuits and no paths of length $> k$. If $\vec{D} \in \Omega(G)$, then the neighbor \vec{H}' of \vec{H} is defined equal to \vec{D} and is optimal since it contains $|E(G)|$ arcs. A new search can therefore be initiated with a smaller value of k . If $\vec{D} \notin \Omega(G)$, a digraph \vec{D}' is built from \vec{D} by adding and orienting an edge that belongs to G but not to D . We know from Property 21 that \vec{D}' is circuit-free but contains paths of length $> k$. Finally, the neighbor $\vec{H}' \in N_4(\vec{H})$ of \vec{H} is obtained from \vec{D}' by removing a minimal (inclusion-wise) set of arcs so that no path in \vec{H}' has length $> k$. The unique arc that is added to \vec{D} to obtain \vec{D}' is introduced in the tabu list TL . This list contains arcs that are not allowed to be removed for several iterations. We have proved in Property 23 that for every $\vec{H} \in \Pi_k(G)$ there is a sequence $(\vec{H}_0, \dots, \vec{H}_n)$ of digraphs such that $\vec{H}_0 = \vec{H}$, $\vec{H}_n \in \Omega(G)$, and each \vec{H}_i belongs to $N_4(\vec{H}_{i-1})$.

At each iteration of the above tabu search algorithms, we set the tabu length $|TL|$ equal to $\lceil \sqrt{x} \rceil$, where x is the number of candidate neighbors at the current solution (see Table 3.3 in the next section).

All these algorithms have been implemented using a very simple approach. Given a digraph \vec{G} , we compute the length $d_{\vec{G}}^+(v)$ of the longest path starting at v and the length $d_{\vec{G}}^-(v)$ of the longest path ending at v , for all vertices $v \in V(G)$. This can be done in $O(|E(G)|)$ time since the search spaces contain only circuit-free digraphs (see for example [AMO93]). Hence, $\lambda(\vec{G})$ is then equal to the largest value $d_{\vec{G}}^+(v)$, and an arc $u \rightarrow v$ lies on a longest path if and only if $d_{\vec{G}}^-(u) + d_{\vec{G}}^+(v) = \lambda(\vec{G})$. In order to evaluate a neighbor solution, we simply perform the required changes and determine the new longest paths in the neighbor solution. We have not worked on efficient implementations which can perhaps make it possible to evaluate neighbor solutions in a shorter time. This could be the topic of another paper where these neighborhoods could be used in conjunction with other ones in a local search coloring algorithm. To summarize, with our implementation of the four algorithms, we find a best neighbor in $O(|E(G)| |N_i(\vec{G})|)$ time. This corresponds to about 1 second CPU-time for graphs with 100 vertices on an AMD Athlon 1.8GHz with 256M DDR memory.

3.6 Computational experiments and conclusions

The objective of the computational experiments is to compare the four algorithms described in the last section. More specifically, we would like to have indications about the relative efficiency of the different neighborhoods. Hence, the computational experiments are not meant to be exhaustive, but rather indicative of the behavior of the algorithms. These results are nonetheless very useful, as they will help to orient future research on the development of more elaborate algorithms.

To test the four algorithms on a graph G , we consider the circuit-free orientation \vec{G}_0 of G obtained by labelling the vertices of G from 1 to $|V(G)|$, and by orienting each edge (u, v) from u to v if and only if u has a smaller label than v . TABU_1, TABU_2 and TABU_3 are all run starting from \vec{G}_0 , until a stopping criterion is met (see here below). We report the smallest length of the longest path ever encountered during the search.

The test procedure is a little bit different for TABU_4. We first fix $k = \lambda(\vec{G}_0) - 1$ and construct a digraph \vec{H} by removing from \vec{G}_0 a minimal (inclusion-wise) subset of arcs so that $\lambda(\vec{H}) = k$. If TABU_4 is able to produce a digraph $\vec{G} \in \Omega(G)$ before meeting the stopping criterion, then we decrease k by one unit, we build a new solution \vec{H} by removing from \vec{G} a minimal (inclusion-wise) subset of arcs so that $\lambda(\vec{H}) = k$, and we pursue the search from \vec{H} . We report the smallest value k for which TABU_4 was able to produce a solution $\vec{H} \in \Omega(G)$ with $\lambda(\vec{H}) = k$.

We have performed tests with two different stopping criteria. In our first experiments, we stop the search as soon as the best neighbor is not strictly better than the best known solution s^* . Our four tabu search algorithms are then descent methods that stop as soon as a local optimum is reached. We report results for ten random graphs with edge density $d = 0.5$. These graphs are obtained by linking a pair of vertices by an edge with probability 0.5, independently for each pair. It is known in the graph coloring community that random graphs with $d = 0.5$ are hard to color. Five of our instances have 50 vertices and are labelled $G_{50,i}$ ($i = 1, \dots, 5$) while the five other graphs have 100 vertices and are labelled $G_{100,i}$ ($i = 1, \dots, 5$).

Graph	TABU_1	TABU_2	TABU_3	TABU_4
$G_{50,1}$	23	21	20	27
$G_{50,2}$	24	22	22	32
$G_{50,3}$	25	20	20	32
$G_{50,4}$	23	19	19	34
$G_{50,5}$	23	19	19	27
$G_{100,1}$	48	43	42	60
$G_{100,2}$	44	42	40	59
$G_{100,3}$	45	40	39	61
$G_{100,4}$	45	41	41	53
$G_{100,5}$	46	39	38	59

Table 3.1: Descent algorithms on ten random graphs.

Graph	TABU_1	TABU_2	TABU_3	TABU_4	<i>Tabucol</i>
$G_{50,1}$	13	19	10	13	10
$G_{50,2}$	13	16	10	13	9
$G_{50,3}$	14	16	10	13	10
$G_{50,4}$	13	14	9	12	9
$G_{50,5}$	12	15	9	12	9
$G_{100,1}$	27	34	16	25	15
$G_{100,2}$	28	38	16	25	16
$G_{100,3}$	27	36	16	25	15
$G_{100,4}$	27	32	15	25	15
$G_{100,5}$	26	34	16	26	15

Table 3.2: Tabu search algorithms on ten random graphs.

The results appear in Table 3.1.

In Table 3.2, we report results on the same ten graphs, but by stopping the algorithms only once a total of 100,000 iterations have been performed. As a point of comparison, we also indicate the number of colors used by *Tabucol*, which is considered as one of the most simple and efficient tabu search coloring algorithm [GH06].

In Table 3.3, we report some statistics. In the first line, we indicate the average size of the neighborhoods. This information is used for fixing the tabu list length (see previous section). So, for example, when using TABU_1 on random graphs $G_{100,i}$ ($i = 1, \dots, 5$), there are on average 110 arcs on longest paths at each itera-

	Random graphs $G_{50,i}$				Random graphs $G_{100,i}$			
	TABU_1	TABU_2	TABU_3	TABU_4	TABU_1	TABU_2	TABU_3	TABU_4
Neighborhood size	56.6	53.8	67.8	6.2	110.0	137.0	182.2	4.7
arcs reverted	1	1.03	8.96	-	1	1.04	8.54	-
arcs removed	-	-	-	0.97	-	-	-	0.94
improving moves	11.4%	0.8%	1.6%	40.8%	10.2%	2.7%	4.2%	35.8%
deteriorating moves	10.5%	0.0%	0.0%	29.8%	7.7%	0.0%	0.0%	23.7%

Table 3.3: Statistics on random graphs.

Graph	$ V $	$ E $	TABU_1	TABU_2	TABU_3	TABU_4	<i>Tabucol</i>
david	87	406	12	12	11	11	11
huck	74	301	11	11	11	11	11
jean	80	254	10	10	10	10	10
queen5_5	25	160	7	9	5	6	5
queen6_6	36	290	9	12	7	9	7
queen7_7	49	476	11	15	8	11	7
queen8_12	96	1368	18	24	13	17	12
queen8_8	64	728	13	20	10	12	9
queen9_9	81	2112	16	22	10	16	10
queen10_10	100	2940	20	24	12	17	11
myciel3	10	20	4	4	4	4	4
myciel4	23	71	5	5	5	5	5
myciel5	47	236	6	6	6	6	6
myciel6	95	755	9	7	7	7	7
mug88_1	88	146	4	4	4	4	4
mug88_25	88	146	4	4	4	4	4
mug100_1	100	166	4	4	4	4	4
mug100_25	100	166	4	4	4	4	4
1-Insertions_4	67	232	5	5	5	5	5
2-Insertions_3	37	72	4	4	4	4	4
3-Insertions_3	56	110	4	4	4	4	4
4-Insertions_3	79	156	4	4	4	4	3
1-FullIns_3	30	100	4	4	4	4	4
1-FullIns_4	93	593	7	5	5	5	5
2-FullIns_3	52	201	5	5	5	5	5
3-FullIns_3	80	346	6	6	6	6	5

Table 3.4: Tabu search algorithms on DIMACS benchmark graphs.

tion, and the tabu list length is on average equal to 11. The second line indicates the number of arcs which change their orientation when moving from a solution to a neighbor one. Hence, this number is always equal to 1 for TABU_1. For TABU_4, this does not apply, and we indicate in this case the average number of arcs that are removed in order to avoid too long paths. In the two last lines, we give the percentage of strictly improving moves, and the percentage of strictly deteriorating moves performed by our algorithms. As shown by Property 16, this last percentage is necessarily equal to zero for TABU_3.

In Table 3.4, we compare the tabu search algorithms on DIMACS benchmark graphs taken from [JT93] or <http://mat.gsia.cmu.edu/COLOR04>, with the second stopping criterion (i.e., a total of 100,000 iterations). For each graph, we also indicate its number of vertices (column labelled $|V|$) and its number of edges (column labelled $|E|$).

It clearly appears that TABU_2 is not competitive at all. While the first local optimum reached by TABU_2 is among the best ones, when compared to the three

other algorithms (see Table 3.1), it seems difficult to reach better regions of the search space (see Tables 3.2 and 3.4). This is perhaps due to the fact that most of the moves performed by TABU_2 (more than 97%) do not change the solution value (see Table 3.3).

TABU_1 and TABU_4 give similar results, with a slight advantage for TABU_4. The best tabu search with an edge orienting strategy is indisputably TABU_3. Notice that TABU_3 is the unique tabu search algorithm studied in this paper for which the search space is not connected (see the example on Figure 3.3). This success is perhaps linked with the fact that the value of a neighbor is never higher than that of the current solution (Property 16). Hence, as shown in Table 3.3, all moves are either improving or do not change the solution value. Notice that we have already observed that TABU_2 (the worst algorithm) also never uses deteriorating moves. Hence, both TABU_2 and TABU_3 spend most of the time in moving in plateaus of the search space. However, the number of arcs that are modified from one iteration to the other is much higher for TABU_3 (see Table 3.3), and this can explain why TABU_3 has more success in finding an improving move towards another plateau of lower altitude.

Thirty-six graphs with at most 100 vertices is of course a small test set for comparing heuristic methods. We think however that these limited experiments give a clear indication on which strategies seem to be the most promising. For the *total orienting strategy*, the reversal of arcs on longest paths between successive colors clearly gives better results than the reversal of a single arc on a longest path. Also, the reversal of a set of arcs on longest paths having all the same tail or the same head does not seem to be an efficient strategy. The unique tabu search algorithm that uses the *k-fixed partial orienting strategy* is TABU_4, and we have shown that it is ranked second among the four tested algorithms.

More experiments of course still have to be performed, and the combination of an edge orienting strategy with a more classical graph coloring algorithm would also be an interesting research subject. This will be the topic of future research based on the theoretical and experimental results contained in this paper.

CHAPITRE 4

MODÈLES DE PROGRAMMATION MATHÉMATIQUE POUR LE PROBLÈME DE LA COLORATION PAR BANDE

La coloration par bande est généralement introduite dans la littérature comme étant simplement un problème de coloration standard auquel on ajoute des contraintes supplémentaires sur la différence entre les couleurs des noeuds adjacents. Typiquement, un problème de coloration par bande est représenté par un graphe avec des poids d_{ij} sur ses arêtes, le but étant de donner une couleur à chaque noeud de sorte que la différence entre les couleurs de deux noeuds adjacents soit supérieure ou égale à d_{ij} . Si tous les poids sont identiques (ils valent tous k), alors on peut résoudre le problème de coloration standard sans tenir compte des poids sur les arêtes et ensuite multiplier toutes les couleurs de notre solution par k . En particulier, si $k = 1$, le problème est identique à la coloration standard.

4.1 Orientation d'arêtes et coloration par bande

Au chapitre précédent, nous avons discuté des propriétés théoriques de l'approche par orientation d'arêtes pour colorer les noeuds d'un graphe dans le cadre d'une méthode de recherche locale. Dans cette section, nous présentons un modèle mathématique associé à cette approche, qui s'avère être un problème de flot, suivi d'une série de manipulations menant à une formulation standard de coloration par bande. Nous terminons cette section en énonçant un théorème qui généralise celui de Gallai [Gal68], Roy [Roy67], et Vitaver [Vit62] dans le cadre de la coloration par bande.

Un graphe G peut être représenté par sa matrice d'adjacence symétrique $A = \{a_{ij}\}$, où a_{ij} vaut 1 si l'arête (i, j) existe, et vaut 0 sinon. De même, un graphe orienté \vec{G} peut être représenté par sa matrice d'adjacence $B = \{b_{ij}\}$, où b_{ij} vaut 1 si l'arc (i, j) existe, et vaut 0 sinon. Le graphe orienté \vec{G} est une orientation du

graphe G si et seulement si $b_{ij} + b_{ji} = a_{ij} = a_{ji}$ pour toute paire de noeuds i, j . Les poids sur les arêtes du graphe G peuvent être représentés par une matrice $\{d_{ij}\}$, où $d_{ij} = 0$ si l'arête (i, j) n'existe pas. On utilisera le terme *distance* pour désigner le poids d'une arête.

Soient G , un graphe, et \vec{G} , une orientation sans circuits de G . Il est possible de calculer la longueur du plus long chemin de \vec{G} en résolvant un problème de flot à coût minimum. Pour ce faire, il suffit d'ajouter un noeud source s et un noeud destination t qui sont reliés par des arcs de coût nul et de capacité infinie à tous les noeuds du graphe (s ne possède que des arcs sortants et t ne possède que des arcs entrants). Tous les arcs (i, j) de \vec{G} ont un coût de $-d_{ij}$. La solution optimale de ce problème consiste à faire passer une unité de flot par le chemin le plus long du graphe, et la valeur optimale de l'objectif est égale à la longueur du chemin, multipliée par -1 . Pour clarifier le modèle, on peut changer le signe de l'objectif, le transformer en un maximum, et changer le signe de tous les coûts pour obtenir le modèle linéaire suivant de flot à coût maximum, dénoté *FCM* (Flot à Coût Maximum) :

(FCM)

Max

$$\sum_{i,j \in V} d_{ij} x_{ij}$$

s.à

$$\sum_{j \in V} x_{sj} = 1$$

$$\sum_{i \in V} x_{it} = 1$$

$$x_{sk} - x_{kt} + \sum_{i \in V} (x_{ik} - x_{ki}) = 0 \quad k \in V,$$

$$x_{ij} \leq b_{ij} \quad i, j \in V,$$

$$x_{ij} \geq 0 \quad i \in V \cup \{s\},$$

$$j \in V \cup \{t\}$$

où x_{ij} est la quantité de flot passant du noeud i au noeud j .

Propriété 24. Pour toute orientation \vec{G} du graphe G , la valeur optimale de FCM est supérieure ou égale à la longueur du plus long chemin du graphe.

Démonstration. Si l'orientation \vec{G} ne possède pas de circuits, il est connu que la solution optimale de *FCM* identifie un plus long chemin du graphe en y faisant passer une unité de flot. Si l'orientation \vec{G} possède des circuits, alors la solution faisant passer une unité de flot par le plus long chemin est réalisable pour *FCM*. On a donc que la valeur optimale de *FCM* est supérieure ou égale à la longueur du plus long chemin. \square

Soit $\vec{O}(G)$, l'ensemble des orientations du graphe G . Le modèle suivant, dénoté *MFCM* (Minimum des Flots à Coût Maximum), permet de trouver l'orientation $\vec{G} \in \vec{O}(G)$ qui minimise la longueur du plus long chemin :

(*MFCM*)

Min $_{\vec{G} \in \vec{O}(G)}$ Max

$$\sum_{i,j \in V} d_{ij} x_{ij}$$

s.à

$$\begin{aligned} \sum_{j \in V} x_{sj} &= 1 \\ \sum_{i \in V} x_{it} &= 1 \\ x_{sk} - x_{kt} + \sum_{i \in V} (x_{ik} - x_{ki}) &= 0 \quad k \in V, \\ x_{ij} &\leq b_{ij} \quad i, j \in V, \\ x_{ij} &\geq 0 \quad i \in V \cup \{s\}, \\ &\quad j \in V \cup \{t\}. \end{aligned}$$

Dans le cas particulier où toutes les distances d_{ij} sont égales à 1, alors le modèle représente le problème de coloration par orientation d'arêtes décrit au chapitre précédent. Maintenant que le modèle a été formulé, nous pouvons entreprendre de le simplifier. En particulier, on peut dualiser le problème de flot à coût maximum comme suit :

$$\begin{array}{ll}
\text{Max} & \sum_{i,j \in V} d_{ij} x_{ij} \\
\text{s.à} & \\
& \sum_{j \in V} x_{sj} = 1 \quad \left| \begin{array}{l} u \\ -v \\ -w_k \\ y_{ij} \end{array} \right. \\
& \sum_{i \in V} x_{it} = 1 \\
& x_{sk} - x_{kt} + \sum_{i \in V} (x_{ik} - x_{ki}) = 0 \quad k \in V, \\
& x_{ij} \leq b_{ij} \quad i, j \in V, \\
& x_{ij} \geq 0 \quad i \in V \cup \{s\}, \\
& \quad \quad \quad j \in V \cup \{t\}.
\end{array}$$

Le dual est le suivant :

$$\begin{array}{ll}
\text{Min} & (u - v) + \sum_{i,j \in V} b_{ij} y_{ij} \\
\text{s.à} & \\
& (w_j - w_i) + y_{ij} \geq d_{ij} \quad i, j \in V, \\
& v \leq w_k \leq u \quad k \in V, \\
& y_{ij} \geq 0 \quad i, j \in V.
\end{array}$$

Dans *MFCM*, on peut remplacer le modèle *FCM* par son dual :

$$\begin{array}{ll}
\text{Min}_{\vec{G} \in \vec{O}(G)} & \text{Min} \quad (u - v) + \sum_{i,j \in V} b_{ij} y_{ij} \\
\text{s.à} & \\
& (w_j - w_i) + y_{ij} \geq d_{ij} \quad i, j \in V, \\
& v \leq w_k \leq u \quad k \in V, \\
& y_{ij} \geq 0 \quad i, j \in V.
\end{array}$$

On peut fixer v à 1 sans perdre de généralité (puisque la contrainte correspondante dans le primal est redondante), et isoler les y_{ij} dans les contraintes :

$$\begin{aligned} & \text{Min}_{\vec{G} \in \vec{\mathcal{O}}(G)} \text{Min} \quad (u - 1) + \sum_{i,j \in V} b_{ij} y_{ij} \\ & \text{s.à} \end{aligned}$$

$$\begin{aligned} y_{ij} & \geq d_{ij} - (w_j - w_i) & i, j \in V, \\ y_{ij} & \geq 0 & i, j \in V, \\ 1 \leq w_k & \leq u & k \in V. \end{aligned}$$

La somme dans la fonction économique contient un terme en (ij) et un en (ji) :

$$b_{ij} y_{ij} + b_{ji} y_{ji}$$

Puisque l'objectif est de minimiser ces termes, les variables b_{ij} et b_{ji} ne servent qu'à sélectionner la plus petite valeur entre y_{ij} et y_{ji} en affectant l'une des deux variables à 1 et l'autre à 0 lorsque $a_{ij} = 1$, ou les deux variables à 0 lorsque $a_{ij} = 0$. On peut alors remplacer ces termes par :

$$a_{ij} \min\{y_{ij}, y_{ji}\}$$

Puisque $a_{ij} = 0$ lorsqu'il n'y a pas d'arête entre i et j , et $a_{ij} = 1$ lorsqu'il y a une arête, on peut restreindre la somme sur les arêtes et laisser tomber a_{ij} :

$$\begin{aligned} & \text{Min}_{\vec{G} \in \vec{\mathcal{O}}(G)} \text{Min} \quad (u - 1) + \sum_{(i,j) \in E} \min\{y_{ij}, y_{ji}\} \\ & \text{s.à} \end{aligned}$$

$$\begin{aligned} y_{ij} & \geq d_{ij} - (w_j - w_i) & i, j \in V, \\ y_{ij} & \geq 0 & i, j \in V, \\ 1 \leq w_k & \leq u & k \in V. \end{aligned}$$

Comme le sous-problème est maintenant indépendant de l'orientation \vec{G} , le $\text{Min}_{\vec{G} \in \vec{\mathcal{O}}(G)}$ de l'objectif est redondant et peut être éliminé. De plus, puisqu'on minimise une somme de y_{ij} indépendants les uns des autres, on peut poser sans perte de généralité que $y_{ij} = (d_{ij} - (w_j - w_i))^+$, où $z^+ = \max\{0, z\}$. On peut alors enlever les contraintes correspondantes, et la formulation devient :

4.2 La formulation en nombres entiers

Dans le premier chapitre, nous avons modélisé le problème de coloration de graphes comme suit :

$$\begin{aligned}
 \text{Min} \quad & \sum_{k \in \bar{\chi}} w_k \\
 \text{s.à} \quad & \sum_{k \in \bar{\chi}} x_{ik} = 1 \quad i \in V, \\
 & x_{ik} \leq w_k \quad i \in V, k \in \bar{\chi}, \\
 & x_{ik} + x_{jk} \leq 1 \quad (i, j) \in E, k \in \bar{\chi}, \\
 & x_{ik} \in \{0, 1\} \quad i \in V, k \in \bar{\chi}, \\
 & w_k \in \{0, 1\} \quad k \in \bar{\chi}.
 \end{aligned}$$

Si l'on voulait modéliser le problème de coloration de graphes par bande à partir d'un tel modèle, il faut faire ressortir le fait que l'on cherche à minimiser la plus grande couleur utilisée. Pour ce faire, on peut ajouter une contrainte forçant w_k à être supérieur ou égal à w_{k+1} . Ainsi, une couleur ne peut être utilisée que si toutes les couleurs précédentes le sont aussi. Il reste à ajouter des contraintes sur la différence entre les couleurs de deux noeuds reliés par une arête. La contrainte suivante fait ce travail :

$$\sum_{d \leq k \leq d+d_{ij}} (x_{ik} + x_{jk}) \leq 1 \quad (i, j) \in E, 1 \leq d \leq \bar{\chi}_a - d_{ij}.$$

Contrairement au modèle original, où $\bar{\chi}$ dénote le nombre de couleurs utilisées, nous utilisons $\bar{\chi}_a$ pour représenter la plus grande amplitude disponible pour l'affectation aux noeuds du graphe. Le modèle obtenu est le suivant :

$$\begin{aligned}
\text{Min} \quad & \sum_{1 \leq k \leq \bar{\chi}_a} w_k \\
\text{s.à} \quad & \sum_{1 \leq k \leq \bar{\chi}_a} x_{ik} = 1 & i \in V, \\
& x_{ik} \leq w_k & i \in V, 1 \leq k \leq \bar{\chi}_a, \\
& w_k \geq w_{k+1} & 1 \leq k \leq \bar{\chi}_a - 1, \\
& \sum_{d \leq k \leq d+d_{ij}} (x_{ik} + x_{jk}) \leq 1 & (i, j) \in E, 1 \leq d \leq \bar{\chi}_a - d_{ij}, \\
& x_{ik} \in \{0, 1\} & i \in V, 1 \leq k \leq \bar{\chi}_a, \\
& w_k \in \{0, 1\} & 1 \leq k \leq \bar{\chi}_a.
\end{aligned}$$

Puisque les couleurs sont représentées par des indices, toute couleur apparaissant dans une solution doit avoir été prédéfinie, ce qui fait que ce modèle est peu approprié pour le cas où les couleurs peuvent prendre des valeurs fractionnaires. Toutefois, cette façon de modéliser reste très valable pour le problème de coloration standard car l'ensemble de couleurs $\bar{\chi}$ ne contient que des entiers et est relativement restreint (en pire cas, il est de taille $|V|$). Or, ce n'est plus le cas pour le modèle de coloration par bande, car $\bar{\chi}_a$ est dépendant des poids d_{ij} sur les arêtes. Cette formulation n'est donc pas intéressante si les poids sont le moins grands par rapport à la taille du graphe $|V|$, car le nombre de variables et de contraintes est directement relié à cette valeur. En d'autres mots, la difficulté du problème est dépendante des d_{ij} .

On peut aussi modéliser le problème de coloration de graphes en introduisant une variable binaire x_s pour chaque stable s dans le graphe, valant 1 si tous les noeuds du stable devront partager la même couleur, et valant 0 sinon. Le problème consiste à identifier le plus petit sous-ensemble de stables tels que chaque noeud du graphe ne fait partie que d'un et un seul stable de l'ensemble. Soit S , l'ensemble de tous les stables dans le graphe.

$$\begin{aligned}
\text{Min} \quad & \sum_{s \in S} x_s \\
\text{s.à} \quad & \sum_{s \in S \mid i \in s} x_s = 1 & i \in V, \\
& x_s \in \{0, 1\} & s \in S.
\end{aligned}$$

En général, le nombre de stables dans un graphe est exponentiel par rapport

à la taille du graphe, et donc, le modèle contient un très grand nombre de variables. Mehrotra et Trick [MT96] introduisent un modèle semblable et utilisent une méthode de génération de colonnes pour le résoudre. Il est sûrement possible d'adapter ce modèle à la coloration par bande, mais contrairement au modèle de coloration standard, l'adaptation n'est pas aussi simple que d'ajouter un ensemble de contraintes. En effet, il ne suffit plus d'identifier des noeuds faisant partie d'un stable, car les contraintes de distance sur les arêtes ne seraient pas respectées.

Considérons maintenant le modèle de coloration par bande, dénoté CB , obtenu dans la section précédente. Soit $B = \{b_{ij}\}$, une matrice de variables binaires représentant une orientation \vec{G} du graphe G (voir au début du chapitre). On peut transformer CB en un modèle de programmation linéaire en nombres entiers à l'aide d'une relation d'ordonnement des amplitudes w_i . Pour ce faire, il suffit de poser $w_i \leq w_j$ lorsque $b_{ij} = 1$. Soit $\bar{\chi}_a$, une constante bornant supérieurement le nombre d'amplitude $\chi_a(G)$. On peut transformer la valeur absolue de CB à l'aide des deux contraintes suivantes :

$$\begin{aligned} w_i - w_j + (\bar{\chi}_a + d_{ij})b_{ij} &\geq d_{ij} \quad (i, j) \in E, \\ w_j - w_i + (\bar{\chi}_a + d_{ij})b_{ji} &\geq d_{ij} \quad (i, j) \in E. \end{aligned}$$

On sait que pour toute solution optimale, toute variable w_i satisfait $1 \leq w_i \leq \chi_a(G) \leq \bar{\chi}_a + 1$. On a donc que la contrainte $w_i - w_j \geq -\bar{\chi}_a$ est toujours valide. Observons ce qui se passe lorsque $b_{ij} = 1$ (forcément, $b_{ji} = 0$) :

$$\begin{aligned} w_i - w_j &\geq -\bar{\chi}_a \quad (i, j) \in E, \\ w_j - w_i &\geq d_{ij} \quad (i, j) \in E. \end{aligned}$$

La première contrainte devient redondante et la seconde contrainte force $w_i \leq w_j$, tout en s'assurant que la distance entre w_i et w_j respecte bien d_{ij} . De même, lorsqu'on fixe $b_{ji} = 1$ (et $b_{ij} = 0$), la seconde contrainte devient redondante et c'est la première qui force alors $w_i \geq w_j$, tout en respectant la contrainte de distance.

Le modèle CB peut donc s'écrire comme suit :

$$\text{Min } u - 1$$

s.à

$$w_i - w_j + (\bar{\chi}_a + d_{ij})b_{ij} \geq d_{ij} \quad (i, j) \in E,$$

$$w_j - w_i + (\bar{\chi}_a + d_{ij})b_{ji} \geq d_{ij} \quad (i, j) \in E,$$

$$1 \leq w_k \leq u \quad k \in V,$$

$$b_{ij} \in \{0, 1\} \quad (i, j) \in E.$$

où $\bar{\chi}_a$ est constante qui borne supérieurement $\chi_a(G) - 1$.

Afin de simplifier davantage ce modèle, on peut réécrire la seconde contrainte en remplaçant b_{ji} par $(1 - b_{ij})$ et en distribuant les $(\bar{\chi}_a + d_{ij})$:

$$w_j - w_i + (\bar{\chi}_a + d_{ij})b_{ji} \geq d_{ij}$$

$$\Leftrightarrow w_j - w_i + (\bar{\chi}_a + d_{ij})(1 - b_{ij}) \geq d_{ij}$$

$$\Leftrightarrow w_j - w_i + (\bar{\chi}_a + d_{ij}) - (\bar{\chi}_a + d_{ij})b_{ij} \geq d_{ij}$$

$$\Leftrightarrow w_j - w_i - (\bar{\chi}_a + d_{ij})b_{ij} \geq -(\bar{\chi}_a + d_{ij}) + d_{ij}$$

$$\Leftrightarrow w_i - w_j + (\bar{\chi}_a + d_{ij})b_{ij} \leq (\bar{\chi}_a + d_{ij}) - d_{ij}$$

$$\Leftrightarrow w_i - w_j + (\bar{\chi}_a + d_{ij})b_{ij} \leq \bar{\chi}_a.$$

En combinant cette relation avec la première contrainte du modèle précédent, nous obtenons la formulation suivante, dénotée $CBOA$ (Coloration par Bande avec Orientation d'Arêtes) :

($CBOA$)

$$\text{Min } u - 1$$

s.à

$$d_{ij} \leq w_i - w_j + (\bar{\chi}_a + d_{ij})b_{ij} \leq \bar{\chi}_a \quad (i, j) \in E,$$

$$1 \leq w_k \leq u \quad k \in V,$$

$$b_{ij} \in \{0, 1\} \quad (i, j) \in E.$$

$$\text{Min } (u - 1) + \sum_{(i,j) \in E} \min\{(d_{ij} - (w_j - w_i))^+, (d_{ji} - (w_i - w_j))^+\}$$

s.à

$$1 \leq w_k \leq u \quad k \in V.$$

Puisque $d_{ij} = d_{ji}$, on peut remplacer $\min\{(d_{ij} - (w_j - w_i))^+, (d_{ji} - (w_i - w_j))^+\}$ par $\min\{(d_{ij} - (w_j - w_i))^+, (d_{ij} - (w_i - w_j))^+\}$. Le minimum est atteint lorsque la réduction est maximale, i.e. $\min\{(d_{ij} - (w_j - w_i))^+, (d_{ij} - (w_i - w_j))^+\} = (d_{ij} - \max\{(w_j - w_i), (w_i - w_j)\})^+ = (d_{ij} - |w_i - w_j|)^+$:

$$\text{Min } (u - 1) + \sum_{(i,j) \in E} (d_{ij} - |w_i - w_j|)^+$$

s.à

$$1 \leq w_k \leq u \quad k \in V.$$

Démontrons qu'il existe une solution optimale de ce modèle où $\sum_{(i,j) \in E} (d_{ij} - |w_i - w_j|)^+ = 0$. Soit $\{u^1, w^1\}$, une solution réalisable ayant des termes positifs dans la sommation. Nous allons démontrer qu'il existe une solution réalisable $\{u^2, w^2\}$ de qualité égale ou meilleure et qui réduit la sommation par rapport à la première solution.

Soit une arête $(p, q) \in E$ telle que $w_p^1 \geq w_q^1$ et $d_{pq} - (w_p^1 - w_q^1) > 0$. Posons :

$$w_k^2 = \begin{cases} w_k^1 + d_{pq} - (w_p^1 - w_q^1) & \text{si } w_k^1 \geq w_p^1 \text{ et } k \neq q \\ w_k^1 & \text{sinon.} \end{cases}$$

Pour toute arête $(i, j) \in E$ telle que $w_i^1 > w_j^1$, il y a deux cas possibles :

- Si $w_i^2 = w_i^1$, alors $w_p^1 \geq w_i^1 > w_j^1$ et $w_j^2 = w_j^1$.
- Si $w_i^2 = w_i^1 + d_{pq} - (w_p^1 - w_q^1)$, alors $w_j^2 \leq w_j^1 + d_{pq} - (w_p^1 - w_q^1)$

Pour toute arête $(i, j) \in E$ telle que $w_i^1 = w_j^1$ et $i \neq q$, il y a deux cas possibles :

- Si $w_i^1 = w_p^1$, alors $w_i^2 = w_i^1 + d_{pq} - (w_p^1 - w_q^1)$ et $w_j^2 \leq w_j^1 + d_{pq} - (w_p^1 - w_q^1)$
- Si $w_i^1 \neq w_p^1$, alors $w_j^2 = w_j^1$.

Dans tous les cas, $w_i^2 - w_j^2 \geq w_i^1 - w_j^1$. De plus, $w_p^2 = w_p^1 + d_{pq} - (w_p^1 - w_q^1) = d_{pq} + w_q^1$ et $w_q^2 = w_q^1$. Combinés, cela donne $w_p^2 = d_{pq} + w_q^2$, et alors $d_{pq} - (w_p^2 - w_q^2) = 0$.

Posons $u^2 = u^1 + d_{pq} - (w_p^1 - w_q^1)$. L'objectif de la solution $\{u^1, w^1\}$ est :

$$\begin{aligned} & (u^1 - 1) + \sum_{(i,j) \in E} (d_{ij} - |w_i^1 - w_j^1|)^+ \\ &= (u^1 - 1) + d_{pq} - (w_p^1 - w_q^1) + \sum_{(i,j) \in E \setminus \{(p,q)\}} (d_{ij} - |w_i^1 - w_j^1|)^+ \\ &\geq (u^2 - 1) + \sum_{(i,j) \in E \setminus \{(p,q)\}} (d_{ij} - |w_i^2 - w_j^2|)^+ \\ &= (u^2 - 1) + \sum_{(i,j) \in E} (d_{ij} - |w_i^2 - w_j^2|)^+ \end{aligned}$$

L'objectif de la deuxième solution est donc au moins d'aussi bonne qualité que la première. De plus, pour tout noeud $i \in V$, $w_i^2 \leq w_i^1 + d_{pq} - (w_p^1 - w_q^1) = u^2 + (w_i^1 - u^1) \leq u^2$. La solution $\{u^2, w^2\}$ est donc réalisable. Enfin, puisque $u^2 = u^1 + d_{pq} - (w_p^1 - w_q^1) > u^1$, la sommation de l'objectif de la seconde solution est donc strictement inférieure à celle de la première solution.

En appliquant successivement cette transformation, la sommation s'annulera éventuellement et on obtiendra alors une solution où $|w_i - w_j| \geq d_{ij}$ pour toute arête $(i, j) \in E$. Au lieu de ce processus itératif, on peut directement ajouter la contrainte dans le modèle et éliminer la sommation dans l'objectif comme suit :

(CB)

$$\text{Min } u - 1$$

s.à

$$|w_i - w_j| \geq d_{ij} \quad (i, j) \in E,$$

$$1 \leq w_i \leq u \quad i \in V.$$

Si les variables w_i étaient contraintes à prendre des valeurs entières, nous aurions une formulation de coloration par bande, où w_i est la couleur du noeud i et u est égal au nombre chromatique (par bande) pour toute solution optimale. La Propriété 25 nous permet d'affirmer que si les d_{ij} sont entiers, comme c'est le cas pour le problème de coloration par bande, alors à toute solution optimale de notre

modèle (où les w_i sont fractionnaires) correspond une solution optimale pour le problème de coloration par bande. En d'autres mots, notre modèle permet simplement plus de flexibilité dans les choix des d_{ij} . Dans la prochaine section, nous verrons comment tirer avantage de la relaxation de l'intégralité sur les variables w_i lorsque l'on tente de résoudre le problème avec la programmation en nombres entiers.

Propriété 25. *Si les d_{ij} sont entiers, on peut obtenir une solution entière au problème de coloration par bande à partir de toute solution fractionnaire du modèle CB en prenant le plancher des w_i .*

Démonstration. La preuve se fait par contradiction. Hypothèse : il existe au moins une arête (i, j) telle que $|\lfloor w_i \rfloor - \lfloor w_j \rfloor| < d_{ij}$. Considérons les variables w_i en ordre croissant de valeur, et remplaçons-les graduellement par le plancher de leur valeur. Soit (i^0, j^0) , la première arête pour laquelle l'hypothèse est vérifiée au cours du remplacement graduel. On peut supposer sans perte de généralité que $w_{i^0} \leq w_{j^0}$. On a donc que $w_{j^0} - \lfloor w_{i^0} \rfloor \geq d_{ij}$, mais que $\lfloor w_{j^0} \rfloor - \lfloor w_{i^0} \rfloor \leq d_{ij} - 1$. Or, ceci impliquerait que $w_{j^0} \geq \lfloor w_{i^0} \rfloor + d_{ij}$ et $\lfloor w_{j^0} \rfloor \leq \lfloor w_{i^0} \rfloor + d_{ij} - 1$, ce qui est impossible. L'hypothèse est donc contredite. \square

Puisque le modèle CB permet des solutions fractionnaires, la notion de couleur (un nombre entier) devient obsolète, tout comme la notion de nombre chromatique. Nous introduisons ici quelques nouvelles définitions afin de mieux représenter les solutions de notre modèle.

L'*amplitude* d'un noeud i est sa valeur w_i .

Le *niveau d'amplitude* d'une solution, dénoté u , est la plus grande amplitude parmi ses noeuds.

Une *amplification* d'un graphe G est une affectation d'amplitudes à tous les noeuds de G , et elle est *légitime* (réalisable) si la différence d'amplitudes entre deux noeuds reliés par une arête est au moins aussi grande que la distance sur cette arête.

Le *nombre d'amplitude* d'un graphe G , dénoté $\chi_a(G)$, est le plus petit niveau d'amplitude possible parmi ses amplifications légales.

Dans le cas où tous les d_{ij} sont des entiers, alors il y a une correspondance entre amplitude et couleur, entre amplification légale et coloration légale, et entre nombre d'amplitude et nombre chromatique du problème de coloration par bande. En particulier, si tous les d_{ij} sont égaux à 1, le nombre d'amplitude d'un graphe G est égal à son nombre chromatique. Pour s'en convaincre, il suffit de remplacer les d_{ij} par 1 dans CB pour obtenir une formulation standard de coloration de graphes.

Théorème 1. *Soit G , un graphe ayant des poids d_{ij} sur ses arêtes. Pour toute orientation \vec{G} du graphe, il existe un chemin de longueur supérieure ou égale au nombre d'amplitude $\chi_a(G)$ moins 1.*

Démonstration. La valeur optimale de $MFCM$ correspond au minimum, parmi toutes les orientations possibles du graphe G , de la longueur du plus long chemin. En particulier, la longueur du plus long chemin de \vec{G} est supérieure ou égale à la valeur optimale de $MFCM$, qui est égale à la valeur optimale de CB , dénotée $\chi_a(G) - 1$. □

Corollaire 26. *(Gallai [Gal68], Roy [Roy67], et Vitaver [Vit62]). Pour toute orientation \vec{G} d'un graphe G , il existe un chemin possédant au moins autant de noeuds que le nombre chromatique $\chi(G)$.*

Démonstration. Puisque $d_{ij} = 1$ pour toute arête (i, j) de G , la valeur optimale de $MFCM$ est égale à $\chi_a(G) - 1 = \chi(G) - 1$. Le nombre de noeuds sur le plus long chemin de \vec{G} est égal à 1 plus le nombre d'arcs sur ce plus long chemin, qui est à son tour égal à la longueur du chemin. Cette longueur étant supérieure ou égale à la valeur optimale de $MFCM$, on a donc que le nombre de noeuds est supérieur ou égal à $1 + (\chi(G) - 1) = \chi(G)$. □

Les seules variables entières dans cette formulation sont les b_{ij} , et leur nombre est égal au nombre d'arêtes dans le graphe. La taille du problème n'est donc pas reliée aux d_{ij} , ce qui en fait une formulation beaucoup plus intéressante pour les instances où les d_{ij} sont le moins grands. De plus, lors de la résolution d'un problème, on peut mettre à jour $\bar{\chi}_a$ en cours d'exécution lorsque $u - 1$ diminue en dessous de $\bar{\chi}_a$ puisque la variable u est une borne supérieure sur $\chi_a(G)$.

Il est important de noter que $\bar{\chi}_a$ sert en réalité à borner la plus grande distance entre les amplitudes de chaque paire de noeuds reliés par une arête. Cette distance est au plus égale au nombre d'amplitude moins 1 pour toute solution optimale. Toutefois, il existe des instances où pour toute solution optimale, cette borne n'est jamais atteinte, i.e. pour toute paire de noeuds reliés par une arête, la différence d'amplitude est strictement inférieure au nombre d'amplitude moins 1 (voir Figure 4.1). Pour ces instances, la constante $\bar{\chi}_a$ pourrait donc prendre une valeur inférieure au nombre d'amplitude moins 1 sans pour autant perdre de solutions optimales. Toutefois, il pourrait exister des instances pour lesquelles l'utilisation d'une constante inférieure au nombre d'amplitude moins 1 admettrait des solutions réalisables mais n'admettrait aucune solution optimale du problème.

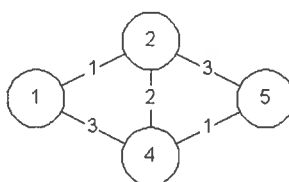


FIG. 4.1 – La différence d'amplitude est toujours inférieure à $\chi_a(G) - 1 = 4$

4.3 La formulation en nombres entiers normalisée

Dans les cas où il serait difficile de trouver une constante $\bar{\chi}_a$ de qualité, nous introduisons une formulation normalisée qui ne requiert pas de déterminer cette constante préalablement. Soit $D = \max_{(i,j) \in E} \{d_{ij}\}$, le plus grand poids sur les

arêtes, et $W = u - 1 + D$, une variable bornant supérieurement le nombre d'amplitude moins 1 plus une distance. On peut remplacer $\bar{\chi}_a + d_{ij}$ par W dans toutes les contraintes pour obtenir le modèle non linéaire suivant :

$$\text{Min } W - D$$

s.à

$$\begin{aligned} w_i - w_j + Wb_{ij} &\geq d_{ij} && (i, j) \in E, \\ w_j - w_i + W(1 - b_{ij}) &\geq d_{ij} && (i, j) \in E, \\ 1 \leq w_k &\leq W + 1 - D && k \in V, \\ b_{ij} &\in \{0, 1\} && (i, j) \in E. \end{aligned}$$

Afin de simplifier le modèle, on remplace w_k par $\bar{w}_k + 1$:

$$\text{Min } W - D$$

s.à

$$\begin{aligned} \bar{w}_i - \bar{w}_j + Wb_{ij} &\geq d_{ij} && (i, j) \in E, \\ \bar{w}_j - \bar{w}_i + W(1 - b_{ij}) &\geq d_{ij} && (i, j) \in E, \\ 0 \leq \bar{w}_k &\leq W - D && k \in V, \\ b_{ij} &\in \{0, 1\} && (i, j) \in E. \end{aligned}$$

Notons que la valeur optimale de ce modèle est $W^* = \chi_a(G) - 1 + D$. Ce modèle n'est pas linéaire, mais il le devient en effectuant le changement de variables suivant :

$$Z = \frac{1}{W} \text{ et } y_i = \frac{\bar{w}_i}{W}.$$

Max Z

s.à

$$\begin{aligned} y_i - y_j + b_{ij} &\geq d_{ij}Z && (i, j) \in E, \\ y_j - y_i + (1 - b_{ij}) &\geq d_{ij}Z && (i, j) \in E, \\ 0 \leq y_k &\leq 1 - DZ && k \in V, \\ b_{ij} &\in \{0, 1\} && (i, j) \in E. \end{aligned}$$

Ce nouveau modèle, dénoté *CBN* (Coloration par Bande Normalisée), est linéaire, et sa valeur optimale Z^* est égale à $\frac{1}{W^*} = \frac{1}{\chi_a(G) - 1 + D}$. On peut donc retrouver $\chi_a(G)$ à partir de Z^* avec la formule suivante : $\chi_a(G) = \frac{1}{Z^*} + 1 - D$. La solution optimale peut aussi être récupérée de manière semblable à partir des y optimaux, dénotés y_i^* , à l'aide de la formule suivante : $w_i = W^* y_i^* + 1 = \frac{y_i^*}{Z^*} + 1$. De plus, si on laisse tomber la contrainte $y_k \leq 1 - DZ$, alors les variables w_i du modèle *CBOA* ne sont plus tenues à être inférieures à u , mais restent bornées dans les contraintes par $\bar{\chi}_a = W + 1 - D$. En particulier, $\bar{\chi}_a^* = W^* + 1 - D$ est une borne inférieure sur la plus petite constante $\bar{\chi}_a$ utilisable dans le modèle *CBOA*. $\bar{\chi}_a^*$ est aussi une borne inférieure sur le nombre d'amplitude moins 1 puisque $\bar{\chi}_a^* \leq \max\{w_i\} - \min\{w_i\} = \chi_a(G) - 1$.

4.4 Résultats préliminaires

Dans des travaux en cours, nous sommes à comparer algébriquement et numériquement les deux modèles (*CBOA* et *CBN*), et à effectuer une comparaison avec d'autres approches pour résoudre le problème, en particulier celle de Prestwich [Pre02b]. Les instances que nous utiliserons pour fins de comparaison proviennent du site internet de la conférence sur la coloration de graphe et ses généralisations : <http://mat.gsia.cmu.edu/COLORING03/>. La taille des instances, appelées GEOM, varie de 20 à 120 noeuds et les poids sur les arêtes varient de 1 à 9.

Les résultats préliminaires de ces comparaisons sont présentés dans les tableaux 4.1 et 4.2, ainsi que dans les tableaux cumulatifs 4.3 et 4.4. Pour évaluer

la robustesse du modèle *CBOA*, nous avons testé quatre valeurs différentes pour la constante $\bar{\chi}_a$; la première valeur est égale à la meilleure valeur connue (fournie par l'algorithme de Prestwich), et les trois autres valeurs sont déterminées subséquemment par incréments de 5 unités. Les versions de *CBOA* ainsi produites sont indicées selon l'incrément utilisé. Par exemple, pour l'instance GEOM20, la valeur obtenue par l'algorithme de Prestwich est de 21. La constante $\bar{\chi}_a$ prend la valeur 21 dans le modèle *CBOA*₀, 26 dans le modèle *CBOA*₅, 31 dans le modèle *CBOA*₁₀, et 36 dans le modèle *CBOA*₁₅. Le but visé par ces incréments est d'évaluer la robustesse du modèle lorsque la qualité de la constante diminue (et par conséquent, la taille de l'espace des solutions augmente). Enfin, puisque le modèle *CBN* n'a pas besoin de constantes, il n'y a qu'une seule version de ce modèle.

Nous avons utilisé CPLEX version 10 pour obtenir des résultats avec trois ensembles différents de paramètres. Le premier ensemble, appelé "Standard", consiste simplement à utiliser les paramètres par défaut du logiciel, et ce pendant une heure. Le second ensemble, appelé "Sans coupes", consiste à utiliser aussi les paramètres par défaut du logiciel, mais sans générer aucune coupe, et ce pendant une heure. Enfin, le troisième et dernier ensemble, appelé "Polishing", consiste à ne résoudre que la racine et puis utiliser l'option "polishing", une heuristique incluse dans le logiciel, pendant une heure. Dans tous les cas, si CPLEX s'arrête avant une heure, c'est qu'une solution optimale a été trouvée (la valeur dans le tableau est marquée d'une étoile). Il arrive parfois qu'une solution entière réalisable de meilleure valeur que celle de Prestwich est trouvée, auquel cas nous avons marqué la valeur d'une croix.

Le premier tableau récapitulatif contient le nombre de fois, parmi les 33 instances, que chaque version de modèle prouve l'optimalité. Le second tableau contient le nombre de fois, parmi les 33 instances, que chaque version de modèle trouve au moins une solution entière réalisable.

À première vue, il semble y avoir une corrélation inverse entre la qualité de la constante du modèle *CBOA* et la capacité à trouver une solution entière réalisable (ou optimale). En effet, plus la constante s'approche de l'optimal, plus il semble

Instance	Prestwich	Paramètres	<i>CBOA</i> ₀	<i>CBOA</i> ₅	<i>CBOA</i> ₁₀	<i>CBOA</i> ₁₅	<i>CBN</i>
GEOM20	21	Standard	21*	21*	21*	21*	21*
		Sans coupes	21*	21*	21*	21*	21*
		Polishing		21*	21*	21*	21*
GEOM20a	20	Standard	20*	20*	20*	20*	20*
		Sans coupes	20*	20*	20*	20*	20*
		Polishing				20*	20*
GEOM20b	13	Standard	13*	13*	13*	13*	13*
		Sans coupes	13*	13*	13*	13*	13*
		Polishing	13*	13*	13*	13*	13*
GEOM30	28	Standard	28*	28*	28*	28*	28*
		Sans coupes	28*	28*	28*	28*	28*
		Polishing		28*	28	28*	28
GEOM30a	27	Standard	27*	27*	27*	27*	27*
		Sans coupes	27*	27*	27*	27*	27*
		Polishing			27	27	27*
GEOM30b	26	Standard	26*	26*	26*	26*	26*
		Sans coupes	26*	26*	26*	26*	26*
		Polishing			26*	26	26
GEOM40	28	Standard	28*	28*	28*	28*	28*
		Sans coupes	28	28	28	28	28*
		Polishing	28*	28	28	28	28
GEOM40a	37	Standard	37	37	37*	37	37
		Sans coupes	37	37	37	37	37
		Polishing					37
GEOM40b	33	Standard	33	33	33*	33	33
		Sans coupes	33	33	33	33	33
		Polishing			33		33
GEOM50	28	Standard	28	28*	28*	28*	28
		Sans coupes	28	28	28	28	28*
		Polishing		28	28	28	28
GEOM50a	50	Standard	50	50	50	50	50
		Sans coupes		50	50	50	50
		Polishing					50
GEOM50b	35	Standard		35	35	35	35
		Sans coupes		35	35	35	35
		Polishing					35
GEOM60	33	Standard	33	33	33	33*	33*
		Sans coupes	33	33	33	33	33
		Polishing		33	33	33	33
GEOM60a	50	Standard		50	51	51	51
		Sans coupes		50	51	51	50
		Polishing					51
GEOM60b	43	Standard		44	45	44	44
		Sans coupes			44	43	44
		Polishing					44
GEOM70	38	Standard	38	38	38*	38*	38
		Sans coupes	38	38	38	38	38*
		Polishing				38	38
GEOM70a	62	Standard		62	62	62	62
		Sans coupes			63	63	61+
		Polishing					62
GEOM70b	48	Standard			52	52	52
		Sans coupes			53	50	51
		Polishing					53

TAB. 4.1 – Résultats complets pour la coloration par bande

Instance	Prestwich	Paramètres	<i>CBOA</i> ₀	<i>CBOA</i> ₅	<i>CBOA</i> ₁₀	<i>CBOA</i> ₁₅	<i>CBN</i>
GEOM80	41	Standard		41	41*	41*	41
		Sans coupes		41	41	41	41*
		Polishing				41	41
GEOM80a	63	Standard			65	65	66
		Sans coupes					64
		Polishing					66
GEOM80b	61	Standard			63	63	63
		Sans coupes				64	64
		Polishing					64
GEOM90	46	Standard	46	46	46	46	46
		Sans coupes	46	46	46	46	46*
		Polishing					46
GEOM90a	64	Standard				69	70
		Sans coupes				72	70
		Polishing					67
GEOM90b	72	Standard				80	78
		Sans coupes				76	77
		Polishing					81
GEOM100	50	Standard		50	50	50	50
		Sans coupes		51	52	50	50
		Polishing					50
GEOM100a	70	Standard					78
		Sans coupes					78
		Polishing					78
GEOM100b	73	Standard					84
		Sans coupes					83
		Polishing					84
GEOM110	50	Standard		51	52	51	51
		Sans coupes			51	52	50
		Polishing					53
GEOM110a	74	Standard					85
		Sans coupes					86
		Polishing					86
GEOM110b	79	Standard					90
		Sans coupes					89
		Polishing					92
GEOM120	60	Standard		59+	61	59+	60
		Sans coupes		61	61	59+	61
		Polishing					61
GEOM120a	84	Standard					99
		Sans coupes					94
		Polishing					96
GEOM120b	87	Standard					100
		Sans coupes					101
		Polishing					100

TAB. 4.2 – Résultats complets pour la coloration par bande (suite)

Paramètres	<i>CBOA</i> ₀	<i>CBOA</i> ₅	<i>CBOA</i> ₁₀	<i>CBOA</i> ₁₅	<i>CBN</i>
Standard	7	8	12	11	8
Sans coupes	6	6	6	6	11
Polishing	2	3	3	4	4
Total (sur 99)	15	17	21	21	23

TAB. 4.3 – Nombre d'instances où l'optimalité est prouvée (sur 33)

Paramètres	<i>CBOA</i> ₀	<i>CBOA</i> ₅	<i>CBOA</i> ₁₀	<i>CBOA</i> ₁₅	<i>CBN</i>
Standard	14	22	25	27	33
Sans coupes	13	19	23	26	33
Polishing	2	5	9	11	33
Total (sur 99)	29	46	57	64	99

TAB. 4.4 – Nombre d’instances où la réalisabilité est atteinte (sur 33)

difficile de trouver une solution au problème. Toutefois, il semble y avoir une corrélation directe entre la qualité de la constante et la qualité des solutions trouvées (lorsque le logiciel y parvient). Donc, en utilisant une constante vraiment mauvaise, on devrait facilement trouver une solution réalisable mais de moindre qualité. C’est dans cette optique que l’on peut mieux analyser les résultats de *CBN*. En effet, on peut considérer *CBN* comme étant une version de *CBOA* qui met à jour sa constante au fur et à mesure que des solutions de meilleure qualité sont trouvées. Ceci explique pourquoi *CBN* trouve toujours une solution entière réalisable pour un problème, tout en obtenant une solution finale de qualité. Les valeurs dans les tableaux semblent montrer une qualité moindre dans les solutions obtenues par *CBN*, mais ces résultats sont biaisés par les instances où *CBN* trouve une solution réalisable alors que les autres modèles n’y arrivent pas. Pour s’en convaincre, il suffit de remarquer que *CBN* est en général meilleur que *CBOA* utilisant comme constante la valeur finale obtenue par *CBN*. En d’autres mots, pour une constante donnée du modèle *CBOA*, on peut supposer que *CBN* pourra trouver une solution de qualité similaire à la constante en question plus rapidement que *CBOA*.

En ce qui concerne l’algorithme de Prestwich, les résultats obtenus par CPLEX y sont identiques jusqu’aux instances de 60 noeuds environ, et c’est seulement à partir des instances de 90 noeuds que l’on peut voir des différences de plus de 5 couleurs entre les solutions des deux approches. Déjà, pour ces instances, le modèle *CBOA* éprouve des difficultés à identifier une solution entière réalisable dans les délais prescrits. Toutefois, il est important de noter que la limite de temps d’une heure n’est peut-être pas appropriée pour résoudre des instances de plus de 100 noeuds, et qu’en utilisant une limite de 5 heures (par exemple), les résultats pourraient être très différents.

4.5 Conclusion

Nous avons entamé le chapitre en introduisant un modèle mathématique de minimisation d'un flot à coût maximum pour représenter le problème de trouver une orientation qui minimise la longueur du plus long chemin d'un graphe ayant des distances sur ses arêtes. Une suite de manipulations mathématiques, incluant la dualisation du sous-problème de flot, nous a mené à une formulation non linéaire de coloration par bande, où les variables représentant les couleurs sont fractionnaires. Le cheminement a aussi servi à introduire une généralisation du théorème de Gallai, Roy et Vitaver pour un graphe avec des distances sur les arêtes.

Ensuite, nous avons transformé le modèle non linéaire de coloration par bande en un modèle de programmation en nombres entiers, dénoté *CBOA*. Nous avons indiqué qu'il est avantageux d'utiliser un tel modèle surtout lorsque les distances sont grandes, car la taille du modèle est indépendante des distances sur les arêtes du graphe. Toutefois, ce modèle nécessite la détermination d'une constante permettant de borner les contraintes. Quelques changements de variables ont suffi pour produire une autre formulation, dénotée *CBN*, qui, cette fois, ne nécessite pas de constante car c'est une variable dans l'objectif qui sert à borner les contraintes.

Les résultats préliminaires tendent à démontrer que ces modèles, en particulier *CBN*, peuvent résoudre le problème de coloration par bande de manière compétitive avec les algorithmes existants pour des instances de moins de 100 noeuds, dans des délais raisonnables.

CONCLUSION

Nous avons présenté une méthode de décomposition qui permet d'améliorer toute fonction de borne supérieure sur la taille de la plus grande clique d'un graphe, et qui fournit comme sous-produit un sous-graphe induit ayant une grande probabilité de contenir la plus grande clique du graphe. Pour ce faire, notre méthode réduit progressivement la taille du graphe en éliminant les noeuds moins susceptibles de faire partie d'une clique de taille maximum, et en mettant à jour une borne supérieure sur la taille de la plus grande clique. Les résultats démontrent bien l'efficacité de notre méthode, puisqu'en incorporant un algorithme de coloration de graphes, nous obtenons une amélioration moyenne de 60%. Avant la cueillette des résultats, nous avons prévu que l'amélioration obtenue serait inversement proportionnelle à l'efficacité de la méthode incorporée. Les résultats semblent toutefois démontrer le contraire, c'est-à-dire que plus la méthode produisant une borne supérieure est efficace, meilleure est l'amélioration.

Il pourrait être intéressant de tester cette nouvelle borne à l'intérieur d'un algorithme de branch-and-bound afin de couper des branches plus tôt dans la résolution. Pour ce faire, on pourrait d'abord utiliser une heuristique pour calculer une borne inférieure sur la taille de la plus grande clique du graphe G . Puis, on choisit une fonction h calculant une borne supérieure sur la taille de la plus grande clique. On évalue alors $h(G)$, puis on applique notre méthode de décomposition (utilisant h) sur ce graphe. À partir de ces trois valeurs (borne inférieure, $h(G)$, décomposition avec h), on peut obtenir une approximation du taux d'amélioration de notre décomposition par rapport à la fonction h seule (en utilisant la même formule que pour nos résultats expérimentaux). Ensuite, on démarre l'algorithme de branch-and-bound en évaluant le potentiel de chaque branche à l'aide de la fonction h appliquée au sous-graphe associé à la branche. On vérifie alors si le taux d'amélioration approximé serait suffisant pour obtenir une évaluation permettant de couper la branche. Si c'est le cas, alors on applique notre décomposition afin d'améliorer l'évaluation de la branche et possiblement la couper. Sinon, on branche

comme d'habitude. Le but de cette approximation du taux est de réduire le nombre de fois que la décomposition est utilisée, car malgré les bons résultats obtenus, son temps de calcul est non négligeable. On pourrait aussi tester des variantes de notre méthode de décomposition, par exemple, en incorporant plusieurs méthodes donnant une borne supérieure et en les utilisant toutes lors de la mise à jour des bornes de chaque noeud du graphe, tout en ne conservant que le meilleur résultat.

Nous avons aussi étudié l'approche d'orientation des arêtes d'un graphe, afin d'obtenir une borne sur le nombre chromatique, dans le contexte de la recherche locale. Cette approche consiste à trouver l'orientation des arêtes du graphe de sorte que le plus long chemin soit le plus court possible. Le nombre de noeuds sur le plus long chemin d'une orientation optimale correspond au nombre chromatique du graphe. Nous avons étudié l'impact d'inverser, de retirer, ou d'ajouter un ou plusieurs arcs d'une orientation donnée sur la formation des circuits et des plus longs chemins du graphe, afin d'identifier des voisinages permettant d'éviter de créer des circuits (qui rendent difficile le calcul du plus long chemin) ainsi que des plus longs chemins (qui détériorent la qualité de la solution). Nous avons identifié et testé plusieurs stratégies en les incorporant dans un algorithme tabou, afin de donner une idée globale de l'efficacité de nos voisinages. En particulier, nous avons déterminé que le voisinage donnant les meilleurs résultats est le seul où nous avons démontré que l'espace des solutions n'est pas connexe. En d'autres mots, il existe des graphes et des solutions initiales telles qu'il est impossible d'atteindre une solution optimale en partant de la solution initiale et en choisissant itérativement une solution dans le voisinage.

Toutefois, l'utilisation de ce voisinage dans une métaheuristique à voisinage variable serait envisageable. Il serait intéressant de trouver d'autres structures semblables aux composantes connexes qui permettent d'inverser plusieurs arcs à la fois tout en ayant des propriétés similaires, comme l'absence de circuits et la non-détérioration de la qualité de la solution courante. Il existe aussi d'autres propriétés intéressantes à étudier, comme la capacité d'une coupe à briser tous les plus longs chemins du graphe en même temps. Déterminer la plus petite coupe qui possède

cette propriété nous donnerait peut-être des indices sur le genre de structure permettant d'améliorer toute solution.

En écrivant le modèle mathématique du problème d'orientation des arêtes d'un graphe, et après de multiples opérations sur ce modèle, nous obtenons une formulation pour la coloration par bande. Par le fait même, nous avons généralisé le théorème de Gallai, Roy et Vitaver pour les graphes ayant des distances sur les arêtes. Ces transformations algébriques nous ont aussi permis de constater que les variables spécifiant les couleurs des noeuds sont fractionnaires, et seules les variables spécifiant l'orientation des arêtes sont binaires. Le nombre de ces variables est égal au nombre d'arêtes dans le graphe, ce qui en fait une formulation compacte pour la coloration par bande (et par le fait même, pour la coloration standard qui est un cas particulier). Toutefois, l'intérêt pour une telle formulation augmente avec les poids sur les arêtes puisque seuls quelques coefficients changent, contrairement à l'approche standard où le nombre de variables dépend du nombre chromatique (par bande) et donc des poids sur les arêtes. Notre formulation utilise une borne supérieure sur le nombre d'amplitude pour calculer ses coefficients, tout comme un modèle standard utiliserait une borne supérieure sur le nombre chromatique (par bande) pour définir ses variables. Dans les cas où calculer une telle borne serait difficile, nous avons introduit une formulation normalisée pour laquelle une telle valeur n'est pas nécessaire. Des résultats préliminaires nous indiquent que la formulation normalisée semble plus efficace que la formulation précédente, et qu'elle est compétitive avec les algorithmes existants pour des graphes de moins de 100 noeuds.

BIBLIOGRAPHIE

- [AF99] D. ACHLIOPTAS et E. FRIEDGUT. A sharp threshold for k -colorability. *Random Structures and Algorithms*, 14(1):63–70, 1999.
- [AHZ03] C. AVANTHAY, A. HERTZ et N. ZUFFEREY. A variable neighborhood search for graph coloring. *European Journal of Operational Research*, 151:379–388, 2003.
- [AJ05] D. APPLGATE et D. JOHNSON. Dfmax source code in C. A World Wide Web page, novembre 2005. <ftp://dimacs.rutgers.edu/pub/challenge/graph/solvers/>.
- [AK97] N. ALON et M. KRIVELEVICH. The concentration of the chromatic number of random graphs. *Combinatorica*, 17(3):303–313, 1997.
- [AKL02] M. ALLEN, G. KUMARAN et T. LIU. A combined algorithm for graph-coloring in register allocation. In *Computational Symposium on Graph Coloring and Generalizations (COLOR02)*, 100–111, Ithaca, New York, USA, 7-8septembre 2002.
- [AMO93] R.K. AHUJA, T.L. MAGNANTI et J.B. ORLIN. *Network flows : Theory, Algorithms, and Applications*. Prentice-Hall, New Jersey, 1993.
- [Aou06] M. AOUCHICHE. *Comparaison automatisée d'invariants en théorie des graphes*. Thèse de doctorat, École polytechnique de Montréal, 2006.
- [AvHK⁺03] K.I. AARDAL, S.P.M. van HOESEL, A.M.C.A. KOSTER, C. MANNINO et A. SASSANO. Models and solution techniques for frequency assignment problems. *4OR : Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(4):261–317, 2003.
- [Bab94] L. BABEL. A fast algorithm for the maximum weight clique problem. *Computing*, 52:31–38, 1994.
- [BA_dN04] V.C. BARBOSA, C.A.G. ASSIS et J.O. do NASCIMENTO. Two novel evolutionary formulations of the graph coloring problem. *Journal of Combinatorial Optimization*, 8:41–63, 2004.

- [Bar00] V.C. BARBOSA. *An Atlas of Edge-Reversal Dynamics*. Chapman & Hall/CRC, London, UK, 2000.
- [BB06] M. BUDINICH et P. BUDINICH. A spinorial formulation of the maximum clique problem of a graph. A World Wide Web page, août 2006. www.citebase.org/abstract?id=oai:arXiv.org:math-ph/0603068.
- [BBBV97] C.H. BENNETT, E. BERNSTEIN, G. BRASSARD et U. VAZIRANI. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, 1997.
- [BBP02] S. BUSYGIN, S. BUTENKO et P.M. PARDALOS. A heuristic for the maximum independent set problem based on optimization of a quadratic over a sphere. *Journal of Combinatorial Optimization*, 6(3):287–297, 2002.
- [BBPP99] I.M. BOMZE, M. BUDINICH, P.M. PARDALOS et M. PELILLO. The maximum clique problem. In D.-Z. DU et P.M. PARDALOS, éditeurs. *Handbook of Combinatorial Optimization*, volume 4, 1–74. Kluwer Academic Publishers, 1999.
- [BBPR02] I.M. BOMZE, M. BUDINICH, M. PELILLO et C. ROSSI. Annealed replication : A new heuristic for the maximum clique problem. *Discrete Applied Mathematics*, 121:27–49, 2002.
- [BC96] M. BROCKINGTON et J.C. CULBERSON. Camouflaging independent sets in quasi-random graphs. In *[JT93]*, 75–88, 1996.
- [BCMM06] P. BEAME, J. CULBERSON, D. MITCHELL et C. MOORE. The resolution complexity of random graph k-colorability. A World Wide Web page, août 2006. citeseer.ist.psu.edu/beame03resolution.html.
- [BCN87] E. BALAS, V. CHVÁTAL et J. NEŠETRIL. On the maximum weight clique problem. *Mathematics of Operations Research*, 12(3):522–535, 1987.

- [BF06] V.C. BARBOSA et R.G. FERREIRA. On the phase transitions of graph coloring and independent sets. A World Wide Web page, août 2006. citeseer.ist.psu.edu/barbosa02phase.html.
- [BFP01] S. BUTENKO, P. FESTA et P. PARDALOS. On the chromatic number of graphs. *Journal of Optimization Theory and Applications*, 109:51–67, 2001.
- [BJH03] A. Di BLAS, A. JAGOTA et R. HUGHEY. A range-compaction heuristic for graph coloring. *Journal of Heuristics*, 9(6):489–506, 2003.
- [Blo01] I. BLOECHLIGER. A new heuristic for the graph coloring problem. Rapport technique, École Polytechnique Fédérale de Lausanne, Département de Mathématiques Appliquées, 2001.
- [BN98] E. BALAS et W. NIEHAUS. Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems. *Journal of Heuristics*, 4(2):107–122, 1998.
- [BP01] R. BATTITI et M. PROTASI. Reactive local search for the maximum clique problem. *Springerlink Algorithmica*, 29(4):610–637, 2001.
- [BPS00] I.M. BOMZE, M. PELILLO et V. STIX. Approximating the maximum weight clique using replicator dynamics. *IEEE Transactions on Neural Networks*, 11(6):1228–1241, 2000.
- [Bré79] D. BRÉLAZ. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.
- [Bro41] R.L. BROOKS. On colouring the nodes of a network. *Mathematical Proceedings of the Cambridge Philosophical Society*, 37:194–197, 1941.
- [Bro72] J.R. BROWN. Chromatic scheduling and the chromatic number problem. *Management Science*, 19(4):456–463, 1972.
- [BS99] V.C. BARBOSA et J.L. SZWARCFITER. Generating all the acyclic orientations of an undirected graph. *Information Processing Letters*, 72(1-2):71–74, 1999.

- [BT90] L. BABEL et G. TINHOFER. A branch and bound algorithm for the maximum clique problem. *ZOR-Methods and Models of Operations Research*, 34:207–217, 1990.
- [Bud03] M. BUDINICH. Bounds on the maximum clique of a graph. *Discrete Applied Mathematics*, 127(3):535–543, 2003.
- [Bus06] S. BUSYGIN. A simple clique camouflaging against greedy maximum clique heuristics. A World Wide Web page, août 2006. cite-seer.ist.psu.edu/busygin02simple.html.
- [BY99] S. BENSON et Y. YE. Approximating maximum stable set and minimum graph coloring problems with the positive semidefinite relaxation. Working paper, novembre 1999. Department of Management Science, University of Iowa, Iowa.
- [BZ03] I. BLOEHLIGER et N. ZUFFEREY. A reactive tabu search using partial solutions for the graph coloring problem. Rapport technique, Institut de Mathématiques, École Polytechnique Fédérale de Lausanne, 2003.
- [CDS03] M. CHIARANDINI, I. DUMITRESCU et T. STUETZLE. Local search for the graph colouring problem. a computational study. Rapport technique AIDA-03-01, FG Intellektik, TU Darmstadt, janvier 2003.
- [CG01] J. CULBERSON et I. GENT. Frozen development in graph coloring. *Theoretical Computer Science*, 265:227–264, 2001.
- [CH97] D. COSTA et A. HERTZ. Ants can colour graphs. *Journal of the Operational Research Society*, 48:295–305, 1997.
- [CHD95] D. COSTA, A. HERTZ et O. DUBUIS. Embedding a sequential procedure within an evolutionary algorithm for coloring problems in graphs. *Journal of Heuristics*, 1:105–128, 1995.
- [CHdW87] M. CHAMS, A. HERTZ et D. de WERRA. Some experiments with simulated annealing for coloring graphs. *European Journal of Operational Research*, 32(2):260–266, 1987.

- [Chi06] M. CHIARANDINI. Bibliography on graph coloring. A World Wide Web page, août 2006. www.imada.sdu.dk/~marco/gcp/.
- [CL96] J.C. CULBERSON et F. LUO. Exploring the k -colorable landscape with iterated greedy. In *[JT93]*, 245–284, 1996.
- [CNP03] V. CUTELLO, G. NICOSIA et M. PAVONE. A hybrid immune algorithm with information gain for the graph coloring problem. *Lecture Notes in Computer Science*, 2723:171–182, 2003.
- [CO06] A. COJA-OGHLAN. The lovasz number of random graphs. A World Wide Web page, août 2006. www.citebase.org/abstract?id=oai:arXiv.org:math/0306266.
- [CP90] R. CARRAGHAN et P.M. PARDALOS. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9:375–382, 1990.
- [CS02] M. CHIARANDINI et T. STUETZLE. An application of iterated local search to graph coloring. In D.S. JOHNSON, A. MEHROTRA et M.A. TRICK, éditeurs. *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations (Ithaca, New York, USA)*, 112–125, 2002.
- [Cul06] J. CULBERSON. Graph coloring resources. A World Wide Web page, août 2006. web.cs.ualberta.ca/~joe/Coloring/index.html.
- [DC99] M. DORIGO et G.D. CARO. The ant colony optimization meta-heuristic. In D. CORNE, M. DORIGO et F. GLOVER, éditeurs. *New Ideas in Optimisation*, 11–32. McGraw-Hill, 1999.
- [Dem79] R.W. DEMING. Acyclic orientations of a graph and chromatic and independence numbers. *Journal of Combinatorial Theory B*, 26:101–110, 1979.
- [DGH04] C. DESROSIERS, P. GALINIER et A. HERTZ. Efficient algorithms for finding critical subgraphs. *Les cahiers du GERAD*, G-2004-31, 2004. À paraître dans *Discrete Applied Mathematics*.

- [DH98] R. DORNE et J.K. HAO. A new genetic local search algorithm for graph coloring. *Lecture Notes in Computer Science*, 1498:745–754, 1998. www.info.univ-angers.fr/pub/hao.
- [EDM03] L. EIN-DOR et R. MONASSON. The dynamics of proving uncolourability of large random graphs : I. Symmetric Colouring Heuristic. *Journal of Physics A : Mathematical and General*, 36(43):11055–11067, octobre 2003.
- [Epp01] D. EPPSTEIN. Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction. In *12th ACM-SIAM Symposium on Discrete Algorithms*, 462–470, 2001.
- [Fah02] T. FAHLE. Simple and fast : Improving a branch-and-bound algorithm for maximum clique. In R. MÖRING et R. RAMAN, éditeurs. *Proceedings of the 10th Annual European Symposium (ESA2000)*, 485–498, Rome, Italy, 2002.
- [FF96] C. FLEURENT et J.A. FERLAND. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63:437–461, 1996.
- [FH00] N. FUNABIKI et T. HIGASHINO. A minimal-state processing search algorithm for graph colorings problems. *IEICE Transaction Fundamentals*, E83-A(7):1420–1430, 2000.
- [FH02] A. FABRIKANT et T. HOGG. Graph coloring with quantum heuristics. In *Eighteenth National Conference on Artificial Intelligence*, 22–27, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.
- [FL97] R. FILHO et G. LORENA. A constructive genetic algorithm for graph coloring. Presented at APORS'97 conference, Melbourne /Australia., 1997. citeseer.ist.psu.edu/filho97constructive.html.
- [FLS01] D.A. FOTAKIS, S.D. LIKOTHANASSIS et S.K. STEFANAKOS. An evolutionary annealing approach to graph coloring. In E.J.W. BOERS, S. CAGNONI, J. GOTTLIEB, E. HART, P.L. LANZI, G. RAIDL, R.E.

- SMITH et H. TIJINK, éditeurs. *Applications of Evolutionary Computing EvoWorkshops2001 : EvoCOP, EvoFlight, EvoIASP, EvoLearn, and EvoSTIM*, volume 2037, 120–129, Como, Italy, 18-19 2001. Springer-Verlag.
- [Gal68] T. GALLAI. On directed paths and circuits. In P. ERDÖS et G. KATONNA, éditeurs. *Theory of Graphs*, 115–118. Academic Press, New York, 1968.
- [GH99] P. GALINIER et J.-K. HAO. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999. www.info.univ-angers.fr/pub/hao.
- [GH01] M.U. GERBER et A. HERTZ. A transformation which preserves the clique number. *Journal of Combinatorial Theory*, 83(2):320–330, 2001.
- [GH06] P. GALINIER et A. HERTZ. A survey of local search methods for graph coloring. *Computers and Operations Research*, 33(9):2547–2562, 2006.
- [GHZ03] P. GALINIER, A. HERTZ et N. ZUFFEREY. An adaptive memory algorithm for the k -colouring problem. *Les cahiers du GERAD*, G-2003-35, 2003.
- [GJ79] M.R. GAREY et D.S. JOHNSON. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, NY, 1979.
- [GK03] F. GLOVER et G. KOCHENBERGER. *Handbook of Metaheuristics*, volume 57 de *International Series in Operations Research & Management Science*. Kluwer Academic, Boston/Dordrecht/London/, 2003.
- [GL97] F. GLOVER et M. LAGUNA. *Tabu Search*. Kluwer Academic Publishers, 1997.
- [GLC04] A. GROSSO, M. LOCATELLI et F. DELLA CROCE. Combining swaps and node weights in an adaptive greedy approach for the maximum clique problem. *Journal of Heuristics*, 10(2):135–152, 2004.

- [GLM03] F. GLOVER, M. LAGUNA et R. MARTÍ. Scatter search. In A. GHOSH et S. TSUTSUI, éditeurs. *Advances in evolutionary computing : theory and applications*, 519–537. Springer-Verlag New York, Inc., New York, NY, USA, 2003.
- [Glo89] F. GLOVER. Tabu search – Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.
- [Gol06] M.C. GOLDBARG. Extra-intracellular transgenetic algorithm applied to the graph coloring problem. A World Wide Web page, août 2006. citeseer.ist.psu.edu/639531.html.
- [GPB03] C. GLASS et A. PRUGEL-BENNETT. Genetic algorithm for graph colouring : exploration of Galinier and Hao’s algorithm. *Journal of Combinatorial Optimization*, 7:229–236, 2003.
- [GPB05] C.A. GLASS et A. PRÜGEL-BENNETT. A polynomially searchable exponential neighbourhood for graph colouring. *Journal of the Operational Research Society*, 56(3):324–330, 2005.
- [GPR96] F. GLOVER, M. PARKER et J. RYAN. Coloring by tabu branch and bound. In *[JT93]*, 285–307, 1996.
- [GVL02] J. GONZÁLEZ-VELARDE et M. LAGUNA. Tabu search with simple ejection chains for coloring graphs. *Annals of Operations Research*, 117:165–174, 2002.
- [HdW87] A. HERTZ et D. de WERRA. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987.
- [Her00] A. HERTZ. On a transformation which preserves the stability number. *Yugoslav Journal of Operations Research*, 10(1):1–12, 2000.
- [Her03] A. HERTZ. Applications des métaheuristiques à la coloration des sommets d’un graphe. In M. PIRLOT et J. TEGHEM, éditeurs. *Résolution de problèmes de RO par les métaheuristiques*, chapitre 1, 21–48. Hermès Science Publication, Paris, 2003.

- [HH02a] J.-P. HAMIEZ et J.-K. HAO. Scatter search for graph coloring. *In Selected Papers from the 5th European Conference on Artificial Evolution*, 168–179, London, UK, 2002. Springer-Verlag.
- [HH02b] F. HERRMANN et A. HERTZ. Finding the chromatic number by means of critical graphs. *ACM Journal on Experimental Algorithmics*, 7(10): 1–9, 2002.
- [HH06a] J.-P. HAMIEZ et J.-K. HAO. An analysis of solution properties of the graph. A World Wide Web page, août 2006. citeseer.ist.psu.edu/646137.html.
- [HH06b] J.-P. HAMIEZ et J.-K. HAO. Experimental investigation of scatter search for graph coloring. A World Wide Web page, août 2006. citeseer.ist.psu.edu/655631.html.
- [HK02] M.M. HALLDÓRSSON et G. KORTSARZ. Tools for multicoloring with application to planar graphs and partial k-trees. *Journal of Algorithms*, 42:334–366, 2002.
- [HLS05] P. HANSEN, M. LABBÉ et D. SCHINDL. Set covering and packing formulations of graph coloring : algorithms and first polyhedral results. *Les cahiers du GERAD*, G-2005-76, 2005.
- [HP00] T. HOGG et D. PORTNOV. Quantum optimization. *Information Sciences*, 128:181, 2000.
- [HRS02] J. HARANT, Z. RYJACEK et I. SCHIERMEYER. Forbidden subgraphs and MIN-algorithm for independence number. *Discrete Mathematics*, 256(1-2):193–201, 2002.
- [HSZ05] A. HERTZ, D. SCHINDL et N. ZUFFEREY. Lower bounding and tabu search procedures for the frequency assignment problem with polarization constraints. *4OR : Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 3(2):139–161, 2005.

- [JC98] D.E. JOSLIN et D.P. CLEMENTS. Squeaky wheel optimization. *In Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98), Madison, WI*, 340–346, 1998.
- [JPR00] A. JAGOTA, M. PELILLO et A. RANGARAJAN. A new deterministic annealing algorithm for maximum clique. *In IJCNN '00 : Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, volume 6, 6505, Washington, DC, USA, 2000. IEEE Computer Society.
- [JS01] A. JAGOTA et L.A. SANCHIS. Adaptive, restart, randomized greedy heuristics for maximum clique. *Journal of Heuristics*, 7(6):565–585, 2001.
- [JT93] D.S. JOHNSON et M.A. TRICK, éditeurs. *Cliques, Coloring, and Satisfiability – Second DIMACS Implementation challenge*, volume 26 de *DIMACS – Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 11–13 octobre 1993.
- [JT95] T.R. JENSEN et B. TOFT. *Graph Coloring Problems*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc., 1995.
- [KHN04] K. KATAYAMA, A. HAMAMOTO et H. NARIHISA. Solving the maximum clique problem by k-opt local search. *In SAC '04 : Proceedings of the 2004 ACM symposium on Applied computing*, 1021–1025, New York, NY, USA, 2004. ACM Press.
- [Kho01] S. KHOT. Improved inapproximability results for maxclique, chromatic number and approximate graph coloring. *In FOCS '01 : Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, 600–609, Washington, DC, USA, 2001. IEEE Computer Society.
- [KJ85] M. KUBALE et B. JACKOWSKI. A generalized implicit enumeration algorithm for graph coloring. *Communications of the ACM*, 28(4):412–418, 1985.

- [Klo02] W. KLOTZ. Graph coloring algorithms. Mathematik-Bericht, Clausthal University of Technology, Clausthal, Germany, 2002.
- [Knu94] D.E. KNUTH. The sandwich theorem. *Electronic Journal of Combinatorics*, 1:48, 1994.
- [Kri97] M. KRIVELEVICH. On the minimal number of edges in color-critical graphs. *Combinatorica*, 17(3):401–426, 1997.
- [Kri02a] M. KRIVELEVICH. Coloring random graphs — an algorithmic perspective. In B. CHAUVIN, P. FLAJOLET, D. GARDY et A. MOKKADDEM, éditeurs. *Proceedings of the 2nd Colloquium on Mathematics and Computer Science : Algorithms, Trees, Combinatorics and Probability (MathInfo'2002)*, 175–195, Birkhäuser, Basel, 2002. Springer-Verlag.
- [Kri02b] M. KRIVELEVICH. Sparse graphs usually have exponentially many optimal colorings. *The Electronic journal of Combinatorics*, 9(1):R27, 2002.
- [Kum04] D. KUMLANDER. A new exact algorithm for the maximum-weight clique problem based on a heuristic vertex-coloring and a backtrack search. In *Fourth European Congress of Mathematics*, Stockholm, Sweden, juin 2004.
- [Kum06] D. KUMLANDER. Incomplete solution approach for the maximum clique finding in the real time systems. In V. DEVED, éditeur. *AIA 2006 : Artificial Intelligence and Applications*, 54–81, Innsbruck, Austria, 13-16 février 2006.
- [KV00] M. KRIVELEVICH et V.H. VU. Approximating the independence number and the chromatic number in expected polynomial time. In *Automata, Languages and Programming*, 13–24, 2000.
- [LAL92] P.J.M. Van LAARHOVEN, E.H.L. AARTS et J.K. LENSTRA. Job-shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, 1992.

- [Las01] B. LASS. Orientations acycliques et le polynôme chromatique. *European Journal of Combinatorics*, 22(8):1101–1123, novembre 2001.
- [Law76] E.L. LAWLER. A note on the complexity of the chromatic number problem. *Information Processing Letters*, 5(3):66–67, 1976.
- [Lov79] L. LOVÁSZ. On the shannon capacity of a graph. *IEEE Transactions on Information Theory*, 25(1):1–7, 1979.
- [LR05] M. LAURENT et F. RENDL. Semidefinite programming and integer programming. In K. AARDAL, G. NEMHAUSER et R. WEISMANTEL, éditeurs. *Handbook on Discrete Optimization*, 393–514. Elsevier B.V., décembre 2005.
- [Lue84] D. G. LUENBERGER. *Linear and Nonlinear Programming*. Addison-Wesley, Reading, Massachusetts, Seconde édition, 1984.
- [LZZ03] A. LIM, X. ZHANG et Y. ZHU. A hybrid method for the graph coloring and its related problems. In *MIC2003 : The Fifth Metaheuristics International Conference*, 2003.
- [Mar02] E. MARCHIORI. Genetic, iterated and multistart local search for the maximum clique problem. In *Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2002*, 112–121, London, UK, 2002. Springer-Verlag.
- [MD00] A. MARINO et R.I. DAMPER. Breaking the symmetry of the graph colouring problem with genetic algorithms. In D. WHITLEY, éditeur. *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, 240–245, Las Vegas, Nevada, USA, 8 2000.
- [MDZ06] I. MÉNDEZ-DÍAZ et P. ZABALA. A branch-and-cut algorithm for graph coloring. *Discrete Applied Mathematics*, 154(5):826–847, 2006.
- [Mit02] D.G. MITCHELL. *The resolution complexity of constraint satisfaction*. Thèse de doctorat, University of Toronto, Toronto, Canada, 2002.

- [MMT05] E. MALAGUTI, M. MONACI et P. TOTH. A metaheuristic approach for the vertex coloring problem. Rapport technique, DEIS, Faculty of Engineering, University of Bologna, Bologna, Italy, 2005.
- [MPBG99] A. MARINO, A. PRUGEL-BENNETT et C. GLASS. Improving graph colouring with linear programming and genetic algorithms. *In Proceedings of EUROGEN99*, 113–118, Jyväskylä, Finland, 1999.
- [MPWZ02] R. MULET, A. PAGNANI, M. WEIGT et R. ZECCHINA. Coloring random graphs. *Physical review letters*, 89(26):268701.1–268701.4, 2002.
- [MT96] A. MEHROTRA et M.A. TRICK. A column generation approach for exact graph coloring. *INFORMS Journal on Computing*, 8:344–354, 1996.
- [Nar02] L. NARAYANAN. Channel assignment and graph multicoloring. *In Handbook of wireless networks and mobile computing*, 71–94. John Wiley & Sons, Inc., New York, NY, USA, 2002.
- [NS01] L. NARAYANAN et S.M. SHENDE. Static frequency assignment in cellular networks. *Algorithmica*, 29(3):396–409, 2001.
- [Öst02] P.R.J. ÖSTERGÅRD. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120:195–205, 2002.
- [Pee83] J. PEEMÖLLER. A correction to Brélaz’s modification of Brown’s coloring algorithm. *Communications of the ACM*, 26(8):595–597, 1983.
- [PH06] W. PULLAN et H.H. HOOS. Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, 25:159–185, 2006.
- [PMX98] P.M. PARDALOS, T. MAVRIDOU et J. XUE. The graph coloring problem : A bibliographic survey. *In D.-Z. DU et P.M. PARDALOS, éditeurs. Handbook of Combinatorial Optimization, Vol. 2*, 331–395. Kluwer Academic Publishers, 1998.
- [Pre01] S. PRESTWICH. Local search and backtracking versus non-systematic backtracking. *In Papers from the AAAI 2001 Fall Symposium on*

- Using Uncertainty Within Computation*, 109–115, North Falmouth, Cape Cod, MA, 2-4 novembre 2001. The American Association for Artificial Intelligence, The AAAI Press.
- [Pre02a] S. PRESTWICH. Coloration neighbourhood search with forward checking. *Annals of Mathematics and Artificial Intelligence*, 34(4):327–340, 2002.
- [Pre02b] S. PRESTWICH. Constrained bandwidth multicoloration neighbourhoods. *In Computational Symposium on Graph Coloring and Generalizations*, 126–133, Ithaca, NY, 2002.
- [PS02a] L. PAQUETE et T. STÜTZLE. An experimental investigation of iterated local search for coloring graphs. *In S. CAGNONI, J. GOTTLIEB, E. HART, M. MIDDENDORF et G. RAIDL, éditeurs. Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002 : EvoCOP, EvoIASP, EvoSTim*, volume 2279, 121–130, Kinsale, Ireland, 3-4 2002. Springer-Verlag.
- [PS02b] V. PHAN et S. SKIENA. Coloring graphs with a general heuristic search engine. *In Computational Symposium on Graph Coloring and its Generalizations*, 92–99, Ithaca, NY, USA, 2002.
- [PT06] M. PELILLO et A. TORSELLO. Payoff-monotonic game dynamics and the maximum clique problem. *Neural Computation*, 18(5):1215–1258, 2006.
- [PX94] P.M. PARDALOS et J. XUE. The maximum clique problem. *Journal of Global Optimization*, 4(3):301–328, 1994. <ftp://dimacs.rutgers.edu/pub/challenge/graph/contributed>.
- [Rég03] J.-C. RÉGIN. Solving the maximum clique problem with constraint programming. *In Fifth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'03)*, 166–179, 2003.

- [RLMS00] H. RAMALHINHO-LOURENÇO, O.C. MARTIN et T. STÜTZLE. Iterated local search. Economics Working Papers 513, Department of Economics and Business, Universitat Pompeu Fabra, novembre 2000.
- [Roy67] B. ROY. Nombre chromatique et plus longs chemins d'un graphe. *Revue française d'informatique et recherche opérationnelle*, 1(5):129–132, 1967.
- [Sew96] E.C. SEWELL. An improved algorithm for exact coloring. In *[JT93]*, 359–373, 1996.
- [Sew98] E.C. SEWELL. A branch and bound algorithm for the stability number of a sparse graph. *INFORMS Journal on Computing*, 10(4):438–447, 1998.
- [SF06] C. SOLNON et S. FENET. A study of ACO capabilities for solving the maximum clique problem. *Journal of Heuristics*, 12(3):155–180, 2006.
- [SL02] P. ST-LOUIS. Heuristique d'évaporation de pénalités dans une méthode de décomposition pour trouver la plus grande clique d'un graphe. Mémoire de maîtrise, Université de Montréal, Montréal, Canada, 2002.
- [SLFG06] P. ST-LOUIS, J.A. FERLAND et B. GENDRON. A penalty-evaporation heuristic in a decomposition method for the maximum clique problem. www.iro.umontreal.ca/~gendron/Publications/PUB_0103.pdf, août 2006.
- [Sta73] R.P. STANLEY. Acyclic orientations of graphs. *Discrete Mathematics*, 5:171–178, 1973.
- [SW68] G. SZEKERES et H.S. WILF. An inequality for the chromatic number of a graph. *Journal of Combinatorial Theory*, 4(1):1–3, 1968.
- [Veg99] S.R. VEGDAHL. Using node merging to enhance graph coloring. *SIGPLAN Notices*, 34(5):150–154, 1999.

- [Vit62] L.M. VITAVER. Determination of minimal coloring of vertices of a graph by means of boolean powers of the incidence matrix. *Doklady Akademii Nauk SSSR*147, 758–759, 1962. In Russian.
- [Vo87] K.-P. VO. Graph colorings and acyclic orientations. *Linear and multilinear algebra*, 22:161–170, 1987.
- [VZ00] A. VESEL et J. ZEROVNIK. How good can ants color graphs? *Journal of Computing and Information Technology - CIT*, 8:131–136, 2000.
- [Wal01] C. WALSHAW. A multilevel approach to the graph colouring problem. Rapport technique 01/IM/69, School of Computing and Mathematical Sciences, University of Greenwich, London SE10 9LS, UK, May 2001.
- [Wol98] L.A. WOLSEY. *Integer Programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley-Interscience, 1998.
- [Woo97] D.R. WOOD. An algorithm for finding a maximum clique in a graph. *Operations Research Letters*, 21(5):211–217, 1997.
- [WTC03] R.L. WANG, Z. TANG et Q.P. CAO. An efficient approximation algorithm for finding a maximum clique using hopfield network learning. *Neural Computation*, 15(7):1605–1619, 2003.
- [WWH05] J.P. WATSON, L.D. WHITLEY et A.E. HOWE. Linking search space structure, run-time dynamics, and problem difficulty : A step toward demystifying tabu search. *Journal of Artificial Intelligence Research*, 24:221–261, 2005.
- [ZZ96] J. ZHANG et H. ZHANG. Combining local search and backtracking techniques for constraint satisfaction. In *Thirteenth National Conference on Artificial Intelligence and Eighth Conference on Innovative Applications of Artificial Intelligence*. AAAI-96, 369–374, Portland, OR, 1996. AAAI Press / MIT Press.