UNIVERSITÉ DE MONTRÉAL

UN ALGORITHME CONSTRUCTIF EFFICACE POUR LE PROBLÈME DE COLORATION DE GRAPHE

MOUHAMED MOURCHID ADIO ADEGBINDIN

DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE INFORMATIQUE)

AVRIL 2013

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

UN ALGORITHME CONSTRUCTIF EFFICACE POUR LE PROBLÈME DE COLORATION DE GRAPHE

présenté par : ADEGBINDIN Mouhamed Mourchid Adio

en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquées

a été dûment accepté par le jury d'examen constitué de :

M. GALINIER Philippe, Doct., président

Mme BELLAÏCHE Martine, Ph.D., membre et directrice de recherche

M. HERTZ Alain, Doct. ès Sc., membre et codirecteur de recherche

Mme LAHRICHI Nadia, Ph.D., membre

REMERCIEMENTS

Mes sincères remerciements vont à l'endroit de ma directrice de recherche Martine Bellaïche sans qui ce travail n'aurait jamais vu le jour. Grâce à son aide, sa compréhension, son encadrement à un moment crucial de ma carrière, cette maîtrise restera pour moi une expérience inoubliable.

Je remercie Alain Hertz, mon co-directeur de recherche dont le support et l'expertise ont joué un rôle déterminant dans la réalisation de ce modeste travail. C'est un honneur pour moi d'avoir été son étudiant.

Je tiens à remercier Samuel Pierre avec qui j'ai débuté ma maîtrise. Son expérience et ses précieux conseils m'ont aidé énormément.

J'exprime ma gratitude à toute l'équipe du service aux étudiants de l'École Polytechnique de Montréal et spécialement à Jonathan Pallet, Vanessa Casanovas i Michel, Claudette Fortier et Philippe Razanakolona. Ils font un travail formidable.

Je ne saurais oublier mes camarades de laboratoire, surtout Richard : ses compétitions de nuits blanches se sont avérées très efficaces durant la rédaction du mémoire.

Merci à mon ami Raimi Rufai pour les sages conseils qu'il ne cesse de me prodiguer. Quelle chance de l'avoir connu.

Enfin, je remercie du fond du cœur ma famille et mes amis pour leur amour, leur tendresse et leur soutien inconditionnels.

RÉSUMÉ

Le problème de coloration de graphe consiste à assigner à chaque sommet une couleur de sorte que deux sommets adjacents n'aient pas la même couleur tout en utilisant le nombre minimal de couleur. C'est l'un des problèmes les plus étudiés en optimisation combinatoire en raison de ses multiples applications (la planification des horaires, l'allocation des ressources, etc.) et de la complexité de sa résolution.

De nombreuses méthodes de résolutions ont été proposées pour résoudre le problème de coloration de graphe. Elles peuvent être réparties en trois catégories : les méthodes exactes dont le temps de calcul croît exponentiellement avec le nombre de sommets du graphe, les méthodes constructives qui donnent rapidement une approximation de la solution optimale du problème, et les métaheuristiques qui fournissent de meilleurs résultats mais au prix d'algorithmes plus complexes et plus gourmands en temps de calcul.

Dans le présent mémoire, nous avons conçu un algorithme constructif d'ordre polynomial qui colore le graphe, une couleur à la fois, en priorisant les voisins des voisins des nœuds déjà colorés. Les résultats des tests effectués sur les graphes classiques du banc d'essai démontrent l'efficacité de l'algorithme proposé qui, pour un temps raisonnable, donne des résultats similaires que TABUCOL, la métaheuristique la plus connue.

ABSTRACT

The *Vertex Coloring Problem* (VCP) requires to assign a color to each vertex in such a way that colors on adjacent vertices are different and the number of colors used is minimized. Due to its numerous practical applications (scheduling, resource allocation, etc.) and computational complexity, the VCP is one of the most studied problems in combinatorial optimization.

Several methods have been proposed to solve the VCP. They can be classified in three families: exact approaches whose running time increases exponentially with the size of the graph, greedy algorithms which approximate the optimal solution in a short time and metaheuristic methods which are the best performing algorithms, but are more complex and time consuming.

In this work, we design a polynomial incremental algorithm which colors the graph, one class at a time by favouring the neighbours of neighbours of already colored vertices. Computational results on the set of DIMACS benchmark instances demonstrate the efficiency of the proposed algorithm which gives the same results as the popular metaheuristic TABUCOL in reasonable running time.

TABLE DES MATIÈRES

REMERCIEMENTS III

RÉSUMÉ IV

ABSTRACT V

TABLE DES MATIÈRES VI

LISTE DES TABLEAUX VIII

LISTE DES FIGURES IX

LISTE DES SIGLES ET ABRÉVIATIONS X

CHAPITRE 1 INTRODUCTION 1

1.1 Le Problème de la coloration de graphe 1

1.2 Domaines d'applications 2

CHAPITRE 2 REVUE DE LITTÉRATURE 4

2.1 Les méthodes exactes 4

2.2 Les méthodes constructives 5

2.3 Les métaheuristiques 6

2.3.1 Les Algorithmes de recherche locale 6

2.3.2 Les algorithmes génétiques 7

2.3.3 Les algorithmes hybrides 7

CHAPITRE 3 ALGORITHME PROPOSÉ 9

3.1 Description de l'algorithme 9

3.2 Calcul de la complexité 11

3.3 Particularités de l'algorithme NoN 12

3.4 Exemple illustratif 13

CHAPITRE 4 RÉSULTATS EXPÉRIMENTAUX 15

4.1 Détails de l'implémentation et choix des algorithmes de comparaison 15
4.2 Plan d'expérience 16
4.3 Contribution de chaque composante de l'algorithme proposé 16
4.4 Le temps d'exécution et le multithreading 19
4.5 Positionnement de l'algorithme proposé dans un contexte général 20
CHAPITRE 5 CONCLUSION ET PERSPECTIVES 25
BIBLIOGRAPHIE 27

LISTE DES TABLEAUX

Tableau 4.1 : Contribution de chaque composante de l'algorithme NoN.....17

Tableau 4.2 : Justification de la priorité dans les règles de sélection.....18

Tableau 4.3 : Le temps d'exécution de l'algorithme NoN 20

Tableau 4.4: Résultats obtenus par TABUCOL, DSATUR et NoN sur les graphes DIMACS.....21

LISTE DES FIGURES

Figure 1 : Coloration optimale de G avec 3 couleurs.....2

Figure 2 : Algorithme des voisins des voisins.....11

LISTE DES SIGLES ET ABRÉVIATIONS

VCP	Vertex Coloring Problem
DIMACS	Discrete Mathematics and Theoretical Computer Science

CHAPITRE 1 INTRODUCTION

« Aux âmes bien nées, la valeur n'attend point le nombre des années. ». Voilà une assertion de Corneille qui se vérifie pour la théorie des graphes. Quoique relativement récents, les graphes sont devenus incontournables en mathématiques discrètes et proposent des méthodes simples et puissamment efficaces pour la résolution des problèmes concrets de la vie quotidienne dans des domaines multiples, qu'il s'agisse de déterminer le nombre minimal de caméras de surveillance nécessaires pour la sécurité d'un immeuble, former des équipes de travail en se fondant sur les affinités des participants, trouver le plus court chemin pour le routage dans un réseau de communication, déterminer l'ordre dans lequel un candidat à la présidentielle américaine doit visiter les cinquante états afin de minimiser le trajet, augmenter la recette globale d'une compagnie aérienne en assignant à chaque vol l'avion le plus adapté, etc.

Un graphe est un ensemble de sommets (nœuds, points) reliés entre eux par des arêtes (lignes, traits) symbolisant une certaine relation. Notre objectif de recherche est de concevoir et d'implémenter un algorithme constructif efficace et fiable pour résoudre le problème de la coloration de graphe qui est un classique de la théorie des graphes. En d'autres termes, il s'agit de proposer un algorithme d'ordre polynomial et déterministe qui donne, en un temps de calcul raisonnable, de meilleurs résultats que les méthodes constructives connues.

Après avoir décrit le problème de coloration de graphe et mentionné ses applications pratiques, nous nous proposons tout d'abord de résumer les méthodes de résolution existantes en mettant l'accent sur les plus connues, puis de détailler l'algorithme proposé et enfin de présenter et commenter objectivement les résultats obtenus.

1.1 Le Problème de la coloration de graphe

Soit $G = (V, E)$ un graphe simple non orienté où V représente l'ensemble des sommets et E l'ensemble de ses arêtes. Deux sommets u et v sont *adjacents* si u et v sont reliés par une arête de E. Une *clique* est un ensemble de sommets deux à deux adjacents et un *stable* est un ensemble de sommets deux à deux non-adjacents.

Le problème de coloration de graphe consiste à assigner à chaque sommet une couleur de sorte que deux sommets adjacents n'aient pas la même couleur, tout en utilisant un nombre minimal de couleurs. Ce dernier est le nombre chromatique $\chi(G)$ du graphe *G*.

La figure ci-après illustre la coloration optimale d'un graphe *G* avec $\chi(G) = 3$.

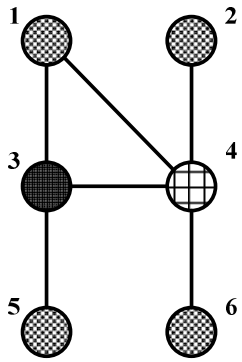


Figure 1 : Coloration optimale de G avec 3 couleurs

1.2 Domaines d'applications

La coloration de graphe est très prisée pour la résolution de problèmes complexes d'optimisation combinatoire. En effet, chaque fois que l'on désire minimiser le nombre de ressources pour effectuer une série de tâches avec la contrainte que certaines tâches ne doivent pas partager la même ressource, il suffit de résoudre le problème de coloration pour le graphe dont les sommets représentent les tâches et où deux sommets sont adjacents si les tâches correspondantes violent la contrainte.

Ainsi, l'allocation des bandes de fréquences dans un réseau cellulaire où des cellules voisines ne doivent pas avoir la même bande de fréquences afin d'éviter les interférences [1], la planification des examens dans une université de sorte qu'un étudiant n'ait pas deux examens au même moment [2], l'organisation d'un tournoi sur la plus courte durée possible en gardant à l'esprit que certaines rencontres ne sauraient avoir lieu simultanément [3], la gestion des ressources dans un univers de *Cloud Computing* [4], sont quelques exemples dans la longue liste des applications pratiques du problème de coloration de graphe.

Pour demeurer compétitif et utilisable, un algorithme de coloration de graphe doit trouver une bonne solution en un temps raisonnable, et ce, pour n’importe quel graphe. C’est pourquoi, comme nous le verrons dans le chapitre suivant, la coloration a fait l’objet de nombreux travaux de recherche.

CHAPITRE 2 REVUE DE LITTÉRATURE

Le problème de coloration de graphe est connu comme étant NP-difficile [5], c'est-à-dire qu'il n'existe pas à ce jour un algorithme polynomial qui donne la coloration optimale pour tout graphe. Dès lors, la coloration de graphe a fait l'objet de nombreuses études résultant en une multitude de méthodes de résolution que l'on pourrait scinder en trois catégories : les méthodes exactes, les méthodes constructives et les métaheuristiques. Nous présentons dans ce chapitre les solutions les plus connues. Pour plus de détails, nous recommandons les revues de littérature de Malaguti et Toth [6] et de Galinier et Hertz [7].

2.1 Les méthodes exactes

La méthode triviale qui vient à l'esprit pour colorer un graphe de n sommets est la méthode exhaustive qui consiste à considérer toutes les k^n assignations possibles pour chaque nombre de couleurs $k = 1, 2, \dots, n$. Même si elle nous donnera à coup sûr le résultat optimal, elle n'est point envisageable lorsque le nombre n de sommets est élevé, le temps d'exécution étant d'ordre exponentiel. C'est d'ailleurs cette impraticabilité, talon d'Achille des méthodes exactes, qui justifie leur petit nombre comparativement aux heuristiques proposées dans la littérature.

Mentionnons cependant l'algorithme de Randall-Brown [8] qui propose de colorer les sommets du graphe à l'aide d'un algorithme d'énumération implicite dans lequel chaque sommet est coloré en utilisant la plus petite couleur possible; Brélaz [9] en 1979 et Peemöller [10] en 1983 ont amélioré cette idée en démarrant l'algorithme par la coloration d'une clique maximale, ce qui donne une borne inférieure sur le nombre chromatique, et en utilisant des critères de sélection pour le prochain sommet à colorer. Mehrotra et Trick [11], en traduisant la coloration de graphe en un problème d'optimisation linéaire en nombres entiers (OLNE), ont développé un algorithme de génération de colonnes plus connu sous le nom de *branch and price*. L'objectif est de minimiser le nombre total de stables nécessaires pour recouvrir tous les sommets du graphe et, en assignant à chaque stable une couleur différente. Cette approche de résolution s'est révélée très intéressante et a servi de tremplin à la plupart des algorithmes exacts ultérieurs [12-15].

2.2 Les méthodes constructives

Les méthodes constructives colorent le graphe un sommet à la fois, en choisissant à chaque étape celui qui semble être le meilleur selon un critère bien défini, sans jamais se remettre en cause. Elles sont très rapides et donnent la plupart du temps une bonne approximation de la solution optimale, mais la qualité du résultat dépend fortement des divers paramètres tel l'ordre de parcours du graphe.

L'algorithme DSATUR [9] développé par Brélaz est incontestablement le plus connu en raison de son efficacité et de sa simplicité. Le principe est le suivant : attribuer la plus petite couleur disponible aux nœuds par ordre décroissant de *degré de saturation* - nombre de couleurs différentes de ses voisins -; en cas d'égalité, prioriser le nœud de degré maximal. L'algorithme s'arrête lorsque tous les sommets sont colorés.

L'autre algorithme, également très populaire, a été proposé la même année par Leighton [16]. Il s'agit de l'algorithme *Recursive Largest First* (RLF) qui construit les classes de couleur, l'une après l'autre - les sommets d'un stable qui ont la même couleur forment une *classe* -; pour chaque couleur *k*, la construction se fait comme suit, avec U_1 représentant l'ensemble des nœuds non colorés qui peuvent recevoir la couleur *k* et U_2 l'ensemble des nœuds non colorés qui ne peuvent plus recevoir la couleur *k* (car ils sont voisins d'au moins un nœud coloré avec la couleur k). Au début de la création de la classe de couleur *k*, U_1 est égal à l'ensemble des sommets qui n'ont aucune des *k*-1 premières couleurs et U_2 est vide. Puis, tant que $U_1 \neq \emptyset$, l'algorithme répète les opérations suivantes :

- Assigner la couleur *k* au nœud *v* de U_1 qui a le plus de voisins dans U_2. En cas d'égalité, choisir un sommet de U_1 qui a le moins de voisins dans U_1. Ôter *v* de U_1.
- Transférer tous les voisins de *v* qui sont dans U_1 vers U_2.

Lorsque la classe de couleur *k* est construite (U_1 est vide), on passe à la couleur *k*+1, en transférant tous les sommets de U_2 vers U_1.

Bollobas et Thomason [17] ont suggéré d'assigner de façon récurrente une couleur au stable de taille maximale dans le graphe formé par les nœuds non colorés, mais cet algorithme peut être très long vu que la recherche du stable de taille maximale dans un graphe est un problème NP-complet [5].

Signalons dans cette section la méthode de Culberson et de Luo [18] qui présente la particularité de remettre en cause les couleurs antérieurement assignées aux nœuds. En fait, ils utilisent de façon itérative un algorithme séquentiel pour colorer le graphe en s'assurant qu'à chaque itération, le nombre de couleurs utilisées ne peut augmenter.

Même si les algorithmes constructifs ne donnent pas les meilleurs résultats, on s'en sert souvent comme point de départ dans les métaheuristiques, ou pour avoir une estimation rapide du nombre chromatique d'un graphe.

2.3 Les métaheuristiques

Par opposition aux méthodes constructives qui partent d'un graphe non coloré et attribuent graduellement une couleur à un sommet ou à un ensemble de sommets pour aboutir à une coloration totale du graphe, les métaheuristiques débutent par une coloration du graphe choisie arbitrairement et tentent d'obtenir une meilleure solution en manipulant la coloration courante.

Cette approche de résolution est plutôt payante car les meilleurs algorithmes actuels pour la coloration de graphe sont indéniablement les métaheuristiques. Elles constituent un domaine phare de recherche pour les théoriciens des graphes et sont, sans surprise, les algorithmes les plus nombreux dans la littérature. On peut les subdiviser en trois types : les algorithmes basés sur la recherche locale, les algorithmes génétiques et les algorithmes hybrides. Nous encourageons le lecteur à consulter les articles de Malaguti et Toth [6] et de Galinier et Hertz [7] pour une revue de littérature plus détaillée.

2.3.1 Les Algorithmes de recherche locale

Les algorithmes de recherche locale sont devenus très prisés depuis la publication du très populaire TABUCOL [19] par Hertz et de Werra en 1987 en s'inspirant de la recherche taboue [20] de Glover. À partir d'une coloration (*solution initiale*) du graphe avec conflits – arêtes ayant la même couleur à leurs extrémités -, TABUCOL tente de minimiser le nombre de conflits (*fonction objectif*) en modifiant la couleur d'un sommet intervenant dans un conflit (*voisinage*). La couleur changée ne peut être assignée au même sommet qu'après un certain nombre d'itérations : cette couleur est dite *taboue* pour ce sommet. La simplicité de TABUCOL et le nouveau type de voisinage suggéré par Morgenstern – *Impasse Class neighborhood* – en 1996 [21] ont contribué au design de meilleurs algorithmes proposés par la suite : *Variable*

Neighborhood Search (VNS) de Avanthay et al. [22], DYN PARTIACOL et FOO- PARTIACOL de Blöchliger et Zuffrey [23], et *Variable Search Space* (VSS) de Hertz et al. [24].

Le recuit simulé est une méthode de recherche locale également très connue. Il se différencie principalement de la recherche taboue par le critère d'acceptation d'une solution dans le voisinage. Ce critère est déterministe pour la recherche taboue mais repose sur un processus probabiliste pour le recuit simulé. Johnston et al. [25] a essayé le recuit simulé sur le problème de coloration de graphe et a proposé, en plus de trois autres implémentations, un algorithme nommé XRLF qui généralise l'algorithme RLF décrit à la section 2.2. En effet, XRLF construit une classe de couleur C en répétant la procédure suivante N1 fois avant de choisir le meilleur résultat. Au départ C est vide et tous les sommets non colorés sont candidats. Si le nombre de candidats est inférieur à une valeur seuil S, on utilise une méthode exacte pour colorer le graphe résiduel. Sinon, Si C est vide, ajouter au hasard un candidat à C et déclarer tous ses voisins non candidats; si C n'est pas vide, choisir dans un sous-ensemble aléatoire de candidats de taille T, le nœud qui a le plus de voisins déclarés non candidats. Les valeurs des paramètres N1, S et T doivent fixés selon le graphe à colorer pour garantir une bonne performance.

2.3.2 Les algorithmes génétiques

Une autre approche de résolution du problème a été suggérée par Davis en 1991 [26]. Il s'agit d'un algorithme purement génétique qui représente une solution par la permutation des sommets et leur attribue de façon séquentielle la première couleur disponible. Les résultats obtenus sur les graphes ne sont pas fameux comparés aux autres métaheuristiques.

2.3.3 Les algorithmes hybrides

La combinaison des algorithmes de recherche locale et des algorithmes génétiques a donné naissance aux algorithmes hybrides ou mémétiques. Ces derniers fournissent de bons résultats sur les graphes au prix d'une complexité sans cesse accrue. Le principe consiste à explorer et diversifier une population de solutions grâce à deux types principaux d'opérateurs : l'opérateur de mutation qui crée de nouvelles solutions en modifiant l'une des solutions initiales à l'aide d'un algorithme de recherche locale et l'opérateur de croisement qui crée de nouvelles solutions en combinant des parties de deux ou de plusieurs solutions existantes (Voir [27]). Citons, entre autres, l'algorithme HCA de Galinier et Hao [28] qui utilise une version améliorée de

TABUCOL et un opérateur de croisement appelé *Greedy Partitioning Crossover*. À partir des solutions parents qui sont des partitions du graphe, l'opérateur *Greedy Partitioning Crossover* considère chaque parent et en extrait la classe de cardinalité maximale pour constituer la solution enfant. En 2008, Malaguti et al. ont proposé le MMT [29] qui adapte l'opérateur de croisement précédent au voisinage suggéré par Morgenstern.

Le principe de l'algorithme à mémoire adaptative AMACOL [30] suggéré par Galinier et al. est un peu différent. Comme son nom l'indique, cette heuristique hybride utilise à chaque itération l'information contenue dans la mémoire pour générer une solution fille. Cette dernière est améliorée à l'aide d'un algorithme de recherche locale et la solution obtenue est utilisée pour mettre à jour la mémoire centrale. Plus récemment, Wu et Hao ont proposé en 2011 l'algorithme EXTRACOL [31] qui commence par extraire du graphe les stables de grande taille avant de colorer le graphe résiduel avec un algorithme mémétique. Les deux auteurs ont poussé plus loin cette façon de procéder dans le tout nouveau IE^{2}COL [32].

CHAPITRE 3 ALGORITHME PROPOSÉ

Il est difficile d'affirmer qu'il existe un *meilleur* algorithme pour la coloration de graphe. L'ordre de complexité des algorithmes exacts les rend impraticables dans la vie réelle où les graphes ont un grand nombre de sommets. Quant aux algorithmes constructifs connus, leur temps d'exécution est rapide mais la qualité des résultats obtenus ne peut égaler celle des métaheuristiques. Ces dernières sont très complexes, indéterministes et exigent souvent le réglage méticuleux d'un grand nombre de paramètres initiaux, selon le graphe. Ce choix forcé entre la simplicité, l'efficacité et ou la rapidité d'exécution a été le leitmotiv de la conception de l'algorithme **NoN** (**N**eighbours **o**f **N**eighbours).

3.1 Description de l'algorithme

Il s'agit d'un algorithme constructif qui colore le graphe, une couleur à la fois, en privilégiant les voisins des voisins des nœuds colorés.

Soit $G = (V, E)$ le graphe à colorer. L'algorithme construit tout d'abord un ensemble stable C_u pour tout sommet u de V, et choisit ensuite parmi ces ensembles stables celui qui constituera la première couleur. Lors de la construction d'un ensemble C_u, on considère le sous-graphe G_{copy} constitué des sommets qui peuvent encore faire partie de C_u. Ce graphe G_{copy} est égal à G au début de la construction de C_u et diminue ensuite jusqu'à devenir le graphe vide, et la construction de C_u prend alors fin. Nous utilisons les notations suivantes :

- V_v est l'ensemble des voisins du nœud v dans le graphe G_{copy}
- W_v est l'ensemble des sommets à distance 2 de v, c'est-à-dire l'ensemble des voisins des voisins du nœud v dans G_{copy} qui ne sont pas voisins de v.
- W_w est l'ensemble des sommets à distance 2 de w

Le premier sommet à être inclus dans C_u est toujours le sommet u. Ensuite, un nouveau sommet w de G_{copy} est rajouté itérativement à C_u sur la base de valeurs que prennent les deux fonctions f et g qui sont définies comme suit :

- la valeur de $f(v)$, au début de la création de l'ensemble C_u, est posée égale au degré du nœud v, pour chaque sommet v de G_{copy}. Ensuite, à chaque fois qu'un sommet w est rajouté à C_u, la valeur $f(v)$ de chaque sommet v de G_{copy} faisant partie de W_w est

augmentée de $|V_v \cap V_w|$ unités. En d'autres termes, on s'intéresse au nombre de voisins que v et w ont en commun dans G_{copy}, et on rajoute cette quantité à $f(v)$.

- la valeur de $g(v)$ au début de la création de l'ensemble C_u, est nulle pour chaque sommet v de G_{copy}. Ensuite, à chaque fois qu'un sommet w est rajouté à C_u, la valeur $g(v)$ de chaque sommet v de G_{copy} faisant partie de W_w est augmentée de $\sum_{i \in V_v \cap V_w} f(i)$ unités. En d'autres termes, pour chaque sommet i voisin à la fois de v et de w, on rajoute la quantité $f(i)$ à $g(v)$.

Règle 1 :

Le sommet w rajouté à C_u est celui qui maximise la fonction g dans G_{copy}; en cas d'égalité, on choisit le nœud qui maximise la fonction f dans G_{copy}.

Règle 2 :

Une fois que tous les ensembles C_u sont construits, on choisit celui pour lequel le nombre d'arêtes du graphe $G - C_u$ est minimal; en cas d'égalité, on choisit celui pour lequel le nombre de sommets de $G - C_u$ est minimal.

Ce stable, forme alors une classe de couleur. L'algorithme retire cette classe de G et recommence la construction d'une nouvelle classe de couleur si le graphe résiduel n'est pas vide. Tout ceci est résumé dans l'algorithme présenté dans la Figure 2.

Initialisation $k = 1$ /* compteur pour les couleurs*/.

1. Tant que $G \neq \{\}$

2. | Pour tout $u \in G$

3. | | $C_u = \{u\}$; $G_{copy} \leftarrow G$; $w \leftarrow u$; $Continue = true$;

| | $\forall\, v \in G_{copy}$, Poser $g(v) = 0$ et $f(v) = \deg(v)$;

4 . | | Tant que $Continue = true$

5. | | | Pour tout $v \in W_w$

6. | | | | $f(v) \leftarrow f(v) + |V_v \cap V_w|$; $g(v) \leftarrow g(v) + \sum_{i \in V_v \cap V_w} f(i)$;

7. | | | Retirer w de G_{copy} ainsi que tous les sommets de V_w .

8. | | | Si $G_{copy} \neq \{\}$, Déterminer le sommet *w* selon la règle (1) et Rajouter w à C_u

9. | | | Sinon $Continue = false$

10. | Déterminer le sommet u selon la règle (2).

| Considérer C_u comme l'ensemble des sommets de couleur *k*

| Supprimer C_u de G.

| $k \leftarrow k + 1$

Figure 2 : Algorithme des voisins des voisins

3.2 Calcul de la complexité

Soit $G = (V, E)$ le graphe à colorer. Les fonctions f et g s'évaluent respectivement en temps constant et linéaire (étape **6**). Avant l'ajout d'un nouveau sommet dans l'ensemble C_u, il est nécessaire de mettre à jour ces deux fonctions pour tout sommet $v \in W_w$ (étape **5**). Autrement dit, le choix de chaque somment de l'ensemble C_u se fait en temps quadratique. En considérant qu'il pourrait y avoir autant d'éléments dans l'ensemble C_u que de sommets dans le graphe G, la construction de chaque ensemble C_u est de l'ordre de $O(n^3)$ et par conséquent, celle de tous les ensembles C_u est de l'ordre de $O(n^4)$. Les étapes précédentes sont répétées pour chaque couleur

utilisée dans le processus. Puisque le nombre de couleurs nécessaires peut être le nombre de sommets du graphe dans le pire des cas (graphe complet dont les sommets sont tous reliés deux à deux par une arête), l'ordre de complexité de l'algorithme NoN est $O(n^5)$.

Mais en réalité, il est de $O(K * n * K_1 * \Delta * n) \approx O(n^2)$ avec K le nombre de couleurs utilisées, K_1 le nombre d'éléments du stable maximal de G (borne supérieure de la cardinalité de l'ensemble C_u) et Δ le degré maximal de $G(\Delta \geq |V_v \cap V_w|)$. C'est ce qui explique son temps d'exécution très court comme nous le verrons par la suite.

3.3 Particularités de l'algorithme NoN

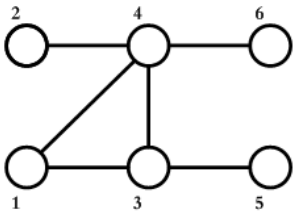
L'algorithme NoN se distingue par un certain nombre de caractéristiques. Il introduit le concept de *lien de parenté* : on part du principe que les nœuds de la même classe de couleur appartiennent à la même *famille*. C'est pourquoi les classes C_u se construisent en déterminant le nœud du graphe résiduel qui a *le lien de parenté* le plus solide – qui maximise les fonctions f et g- avec les nœuds déjà présents dans C_u. Dès lors, la fonction g joue un rôle clé car elle permet de mesurer l'impact de chaque sélection sur les voisins des voisins du sommet coloré.

En outre, contrairement à la plupart des algorithmes constructifs qui colorent le graphe en démarrant par un sommet de départ, l'algorithme NoN exécute une routine gloutonne pour construire un ensemble C_u pour chaque sommet afin d'identifier le meilleur sommet de départ. Cette procédure, bien que relativement coûteuse en temps de calcul, a un effet concret sur la qualité du résultat et confère à l'algorithme son caractère régulier. Avec une telle structure, il est facile d'améliorer considérablement le temps d'exécution de l'algorithme NoN grâce au multithreading, puisque la construction des classes C_u pour chaque sommet u du graphe G sont des opérations complètement indépendantes et peuvent s'exécuter simultanément.

Le thread étant l'unité d'allocation du processeur, un programme monothread ne peut s'exécuter que sur un seul processeur à la fois occasionnant une perte de 50% des ressources sur système à deux processeurs et de 99% sur un système à cent processeurs. Avec le multithreading, la construction des classes C_u peut s'exécuter en parallèle sur plusieurs processeurs afin d'exploiter plus efficacement les ressources CPU disponibles; les systèmes multiprocesseurs étant devenus le standard dans les ordinateurs personnels (souvent quatre processeurs).

3.4 Exemple illustratif

Nous nous proposons de colorer le graphe de la figure 1 avec l'algorithme NoN.



Graphe G

1. $G = \{1, 2, 3, 4, 5, 6\} \neq \{\}$;

2. $u = 1$

3. $C_1 = \{1\}$; $Continue = true$;

v	1	2	3	4	5	6
$g(v)$	0	0	0	0	0	0
$f(v)$	2	1	3	4	1	1

5. $W_1 = \{2, 5, 6\}$

$* v = 2; V_2 \cap V_1 = \{4\}$

$g(2) = 0 + f(4) = 4;$

$f(2) = 1 + |V_2 \cap V_1| = 1 + 1 = 2$

$* v = 5; V_5 \cap V_1 = \{3\}$

$g(5) = 0 + f(3) = 3$

$f(5) = 1 + |V_5 \cap V_1| = 2$

$* v = 6; V_6 \cap V_1 = \{4\}$

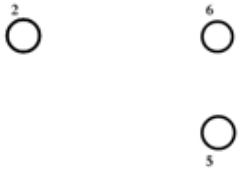
$g(6) = 0 + f(4) = 4$

$f(6) = 1 + |V_6 \cap V_1| = 2$

v	1	2	3	4	5	6
$g(v)$	0	4	0	0	3	4
$f(v)$	2	2	3	4	2	2

7. $G_{copy} = \{2, 5, 6\}$

8. $G_{copy} \neq \{\}$; $w = 2$; $C_1 = \{1, 2\}$;



Graphe **Gcopy**

5. $W_2 = \{\}$

7. $G_{copy} = \{5, 6\}$

8. $G_{copy} \neq \{\}$; $w = 6$; $C_1 = \{1, 2, 6\}$;

5. $W_6 = \{\}$

7. $G_{copy} = \{5\}$

8. $G_{copy} \neq \{\}$; $w = 5$; $C_1 = \{1, 2, 6, 5\}$;

5. $W_5 = \{\}$

7. $G_{copy} = \{\}$

9. $G_{copy} = \{\}$ et $Continue = false$;

2. $u = 2$

3. $C_2 = \{2\}$; $Continue = true$;

v	1	2	3	4	5	6
$g(v)$	0	0	0	0	0	0
$f(v)$	2	1	3	4	1	1

5. $W_2 = \{1, 3, 6\}$

$* v = 1; V_1 \cap V_2 = \{4\}$

$g(1) = 0 + f(4) = 4;$

$f(1) = 2 + |V_1 \cap V_2| = 3$

$* v = 3; V_3 \cap V_2 = \{4\}$

$g(3) = 0 + f(4) = 4;$

$f(3) = 3 + |V_3 \cap V_2| = 4$

* $v = 6; V_6 \cap V_2 = \{4\}$

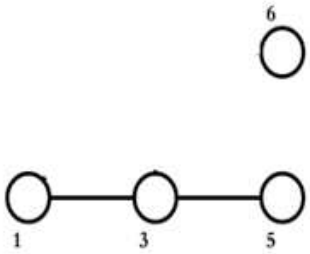
$g(6) = 0 + f(4) = 4;$

$f(6) = 1 + |V_6 \cap V_2| = 2$

v	1	2	3	4	5	6
$g(v)$	4	0	4	0	0	4
$f(v)$	3	1	4	4	1	2

7. $G_{copy} = \{1, 3, 5, 6\}$

8. $G_{copy} \neq \{\}$; $w = 3$; $C_2 = \{2, 3\}$;



Graphe **Gcopy**

5. $W_3 = \{\}$

7. $G_{copy} = \{6\}$

8. $G_{copy} \neq \{\}$; $w = 6$; $C_2 = \{2, 3, 6\}$;

5. $W_6 = \{\}$

7. $G_{copy} = \{\}$

9. $G_{copy} = \{\}$ et $Continue = false$;

Similairement, on obtient :

2. $u = 3$

$C_3 = \{3, 2, 6\}$

2. $u = 4$

$C_4 = \{4, 5\}$

2. $u = 5$

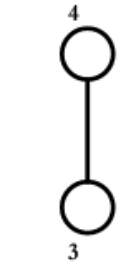
$C_5 = \{5, 4\}$

2. $u = 6$

$C_6 = \{6, 3, 2\}$

10. $C_u = C_1 = \{1, 2, 6, 5\}$

$G - C_1 = \{3, 4\}$; Prochaine couleur



Graphe G

1. $G = \{3, 4\} \neq \{\}$;

2. $u = 3$

$C_3 = \{3\}$

2. $u = 4$

$C_4 = \{4\}$

10. $C_u = C_3 = \{3\}$

$G - C_3 = \{4\}$; Prochaine couleur

1. $G = \{4\} \neq \{\}$;

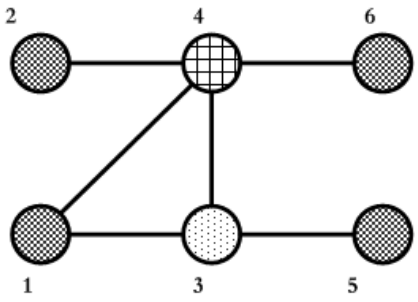
2. $u = 4$

$C_3 = \{4\}$

10. $C_u = C_4 = \{4\}$

$G - C_4 = \{\}$;

1. $G = \{\}$; **Fin**



Graphe G avec 3 couleurs

CHAPITRE 4 RÉSULTATS EXPÉRIMENTAUX

Afin d'évaluer les performances de l'algorithme NoN, nous l'avons implémenté et testé sur les instances des graphes du centre DIMACS (*center for **D**iscrete **M**athematics and **T**heoretical **C**omputer **S**cience*) (voir http://mat.gsia.cmu.edu/COLOR03). Ces graphes constituent le banc d'essai classique pour le problème de coloration et ils ont le double avantage d'avoir des structures très variées et de modéliser des problèmes concrets de la vie réelle. Les résultats sont comparés avec les meilleurs résultats de l'état de l'art et ceux de quelques algorithmes triés sur le volet.

4.1 Détails de l'implémentation et choix des algorithmes de comparaison

L'algorithme NoN a été exécuté sur un ordinateur Fedora 17 équipé d'un processeur i7-3770 et une mémoire RAM de 16 Go. Le langage Java a été choisi pour implémenter l'algorithme NoN pour sa portabilité et surtout parce qu'il facilite le développement de programmes multithreads. Cependant les multiples librairies de Java -non nécessaires pour notre programme - ont un impact sur le temps d'exécution et une implémentation en C++ ou en C serait plus rapide, même si la gestion du multithreading serait plus laborieuse.

Puisque que la grande partie des opérations se basent sur les voisins d'un sommet. Le graphe est conservé dans une table de hachage *HashMap* où la clé est l'indice du sommet et la valeur est un ensemble *HashSet* contenant les indices de tous ses voisins.

Comme mentionné plus haut, la seule partie parallélisable est la construction des ensembles C_u et elle est gérée par un *pool* de threads du framework *Executor* de Java. La taille d'un pool de threads dépend de l'application et de l'environnement de test; après une série d'essais, la taille du pool a été fixée à huit, le nombre de processeurs disponibles. Pour éviter les problèmes typiques des applications concurrentes comme la compétition (*race condition*) et l'interblocage (*deadlocks*), une copie du graphe est nécessaire pour la construction de chaque classe C_u.

Une longue liste d'algorithmes a été proposée pour le problème de coloration de graphe. Afin d'effectuer une comparaison objective, les algorithmes suivants ont été sélectionnés : l'algorithme RLF qui colore aussi le graphe une couleur à la fois, l'algorithme DSATUR et la

métaheuristique la plus populaire TABUCOL. Nos résultats sont également comparés avec les meilleurs résultats de l'état de l'art.

4.2 Plan d'expérience

Afin d'évaluer les performances de l'algorithme NoN, nous avons suivi le plan d'expérience suivant. Tout d'abord, l'apport des différentes composantes de l'algorithme NoN a été mis en évidence de même que la pertinence des règles de sélection 1 et 2. Ensuite, une série de tests a été réalisée pour montrer le temps d'exécution de l'algorithme NoN et souligner l'effet du multithreading. Enfin, une comparaison des résultats de l'algorithme NoN avec les meilleurs résultats de l'état de l'art a été faite dans le but de le positionner dans un contexte plus général.

4.3 Contribution de chaque composante de l'algorithme proposé

Nous rapportons dans le tableau 4.1 les performances de différentes versions de l'algorithme dans le but d'apprécier l'apport de chacune de ses composantes notamment la fonction g et la construction des classes C_u pour tous les sommets. Pour éviter d'encombrer le tableau, quelques instances de graphes jugés représentatifs et difficiles ont été retenues pour cette évaluation : entre autres, les DSJCxxx.x générés aléatoirement en liant une paire de nœuds par une arête avec une certaine probabilité p - leur nombre chromatique n'est pas connu - et les flatxxx_xx_0 dont le nombre chromatique est connu, mais n'ont jamais été colorés optimalement. Les trois premières colonnes indiquent le nom de l'instance, le nombre de sommets et le nombre d'arêtes. Les résultats de l'algorithme RLF (il ressemble le plus à la version simple de l'algorithme NoN), tirés de [33], sont consignés dans la colonne 4. Cet algorithme ressemble beaucoup à l'algorithme NoN dans sa version la plus simple NoN 0.0 - où seule la fonction *f* est utilisée comme critère de comparaison et une classe C_u ayant pour sommet de départ le nœud de degré maximal est construite à chaque étape. Les résultats obtenus se trouvent dans la colonne 5. La colonne 6 montre les résultats obtenus en ajoutant la fonction g comme critère de comparaison (fonction f + fonction g) ; si par contre, on la modifie en construisant toutes les classes C_u, on obtient les résultats de la colonne 7 (fonction *f* + construction de tous les C_u). La dernière colonne exhibe les résultats de NoN dans sa version complète (fonction f + fonction g + construction de tous les C_u). Puisque les résultats de l'algorithme NoN ne changent pas d'une exécution à l'autre, il n'est pas nécessaire de le tester plusieurs fois sur la même instance.

Sur la base des chiffres de la colonne 5, la version simple de l'algorithme donne des résultats moins bien que ceux de RLF : cela est fort prévisible puisqu'il n'y a aucune originalité dans cette version. L'utilisation de la fonction g comme critère de choix améliore clairement les performances de l'algorithme. La différence sur le nombre de couleurs utilisées peut monter jusqu'à 10 couleurs pour le DSJC1000.5 et 11 couleurs pour le flat1000_76_0 (en comparaison avec NON 0.0). On constate également que cette nouvelle version est meilleure que RLF avec un gain moyen de deux couleurs.

Tableau 4.1 : Contribution de chaque composante de l'algorithme NoN

Nom	n	m	RLF	NoN 0.0	Fonction g	Classes u	NoN
DSJC250.5	250	15668	36	38	35	32	31
DSJC250.9	250	27897	83	90	84	76	75
DSJC500.1	500	12458	15	16	15	15	14
DSJC500.5	500	62624	62	66	62	53	51
DSJC500.9	500	224874	155	162	156	135	135
DSJR500.1	500	3555	14	13	13	13	13
DSJR500.1c	500	121275	91	92	89	97	96
DSJR500.5	500	58862	133	134	133	127	128
DSJC1000.1	1000	49629	25	27	25	24	22
DSJC1000.5	1000	249826	110	119	109	95	90
DSJC1000.9	1000	449449	287	307	284	242	239
latin_square_10	900	307350	122	144	134	112	108
le450_15c	450	16680	23	24	23	22	20
le450_15d	450	16750	23	24	23	22	19
le450_25c	450	17343	29	29	28	28	27
le450_25d	450	17425	29	30	27	28	27
flat300_26_0	300	21633	40	42	39	35	33
flat300_28_0	300	21695	40	43	39	35	34
flat1000_50_0	1000	245000	108	116	108	91	88
flat1000_60_0	1000	245830	108	116	108	91	89
flat1000_76_0	1000	246708	109	118	107	91	89

Avec la construction de toutes les classes C_u, la performance de l'algorithme NoN saute à l'œil comme l'indique les résultats de la colonne 7. Cette fois-ci la différence du nombre de couleurs, comparativement à NoN 0.0, est de 24 couleurs pour le DSJC1000.5 par exemple. Autrement dit, même sans l'utilisation de la fonction g, l'algorithme NoN a déjà une grande longueur d'avance sur RLF. Toutefois, Il est important de rappeler que l'efficacité de la fonction g est intimement liée au sommet de départ. Dans le cas présent, on peut affirmer que le sommet de degré maximal

n'est pas le sommet idéal à considérer comme point de départ. L'apport de la fonction g est palpable en observant la colonne 8 car la version complète réduit encore le nombre de couleurs nécessaires pour colorer les graphes considérés. En guise d'illustration, le nombre de couleurs passe de 95 à 90 pour le DSJC1000.5 en considérant la fonction *g*.

Un autre aspect de l'algorithme NoN qui peut influencer sa performance est l'établissement des règles 1 et 2 qui permettent de sélectionner les sommets et la meilleure classe C_u. Puisque chaque règle est constituée de deux critères le tableau 4.2 justifie l'ordre de priorité choisi. Les trois premières colonnes sont les mêmes que dans le tableau précédent. La colonne 4 (colonne 8 du tableau 4.1) montre les résultats de la version complète de NoN où la fonction g est priorisée sur la fonction f pour la sélection du prochain sommet à inclure dans la classe C_u (Règle 1) et le nombre d'arêtes du graphe $G - C_u$ est préféré au nombre de sommets de $G - C_u$ lors du choix de la meilleure classe C_u (Règle 2). La colonne 5 montre les variations observées lorsque l'on intervertit l'ordre de priorité pour la règle 1 tandis que la colonne 6 montre les différences causées par le changement de priorité dans la règle 2.

Tableau 4.2 : Justification de la priorité dans les règles de sélection

Nom	n	m	NoN	fonction f puis fonction g	sommets puis arêtes
DSJC250.5	250	15668	31	31	31
DSJC250.9	250	27897	75	77	75
DSJC500.1	500	12458	14	14	14
DSJC500.5	500	62624	51	52	51
DSJC500.9	500	224874	135	135	135
DSJR500.1	500	3555	13	13	13
DSJR500.1c	500	121275	96	93	93
DSJR500.5	500	58862	128	127	127
DSJC1000.1	1000	49629	22	23	22
DSJC1000.5	1000	249826	90	91	90
latin_square_10	900	307350	108	109	110
le450_15c	450	16680	20	21	20
le450_15d	450	16750	19	21	20
le450_25c	450	17343	27	27	27
le450_25d	450	17425	27	28	28
flat300_26_0	300	21633	33	34	33
flat300_28_0	300	21695	34	34	34
flat1000_50_0	1000	245000	88	89	88
flat1000_60_0	1000	245830	89	90	88
flat1000_76_0	1000	246708	89	90	89

En observant les résultats de la colonne 5, à l'exception des graphes DSJR500.1c et DSJR500.5, le nombre de couleurs augmente en moyenne d'une unité lorsque la fonction f est privilégiée sur la fonction g. Cette légère détérioration de la qualité justifie en partie la préférence de la fonction g sur la fonction f. L'autre raison est un souci de performance : la plupart du temps du temps, la fonction *g* suffit pour départager les sommets.

Les résultats de la colonne 6 sont très similaires à ceux de la colonne 4. On aurait tendance à affirmer que l'ordre de priorité dans la règle 2 n'a pas d'importance mais en réalité il n'en est rien ; les résultats se ressemblent ici parce que le nombre de sommets du graphe résiduel est souvent proportionnel à son nombre d'arêtes. Si l'on prend en considération le fait que le nombre chromatique d'un graphe sans arêtes de 1000 sommets est 1 et celui d'un graphe de deux sommets et une seule arête est 2, prioriser les arêtes sur les sommets semble logique.

4.4 Le temps d'exécution et le multithreading

Il est temps de parler du temps d'exécution de l'algorithme NoN vu qu'il est un critère d'évaluation tout aussi important que la qualité du résultat. Dans le tableau 4.3, nous avons conservé les mêmes instances que précédemment. Une des nouveautés ici est la compilation des résultats de TABUCOL, à savoir le temps de calcul en secondes et le nombre de couleurs, respectivement dans les colonnes 5 et 6. Ces résultats sont tirés de [30] et correspondent à la version dénommée Long_TABU ou LT. Viennent ensuite dans les colonnes 7 et 8, les résultats de RLF qui sont extraits de [33]. Dans les trois dernières colonnes, on indique successivement les résultats de l'algorithme NoN sur chaque instance, le temps de calcul en secondes sans et avec multithreading. Le nombre de threads utilisé est égal à huit.

Nous tenons à rappeler qu'il ne s'agit pas pour nous de comparer le temps de calcul de NoN et de TABUCOL. Ces valeurs sont présentes dans le tableau à titre indicatif et pour un souci de clarté.

D'après le tableau 4.3, l'algorithme NoN est plus lent que les RLF mais il donne de bien meilleurs résultats. Le temps de calcul de NoN augmente avec le nombre de sommets du graphe et est également sensible au nombre d'arêtes. Avec les valeurs de la dernière colonne, on observe qu'en utilisant 8 threads, le temps d'exécution de NoN devient beaucoup plus court. Il est vrai que le gain de temps n'est pas strictement proportionnel au nombre de threads utilisés car

il y a toujours une partie séquentielle à exécuter. Cependant, l'écart est plus perceptible lorsque n est élevé.

Tableau 4.3 : Le temps d'exécution de l'algorithme NoN

Nom	n	M	TABUCOL		RLF		NoN		
			#	T	#	T	#	T (1 thread)	T (8 threads)
DSJC250.5	250	15668	29	19	36	0	31	4	1
DSJC250.9	250	27897	72	-	83	0	75	13	3
DSJC500.1	500	12458	13	-	15	0	14	3	1
DSJC500.5	500	62624	50	173	62	0	51	57	13
DSJC500.9	500	224874	130	-	155	0	135	185	42
DSJR500.1	500	3555	13	-	14	0	13	2	0
DSJR500.1c	500	121275	86	-	91	0	96	97	20
DSJR500.5	500	58862	128	-	133	0	128	96	22
DSJC1000.1	1000	49629	22	-	25	0	22	50	11
DSJC1000.5	1000	249826	89	2534	110	2	90	900	200
DSJC1000.9	1000	449449	245	-	287	7	239	2901	730
latin_square_10	900	307350	106	-	122	4	108	812	186
le450_15c	450	16680	16	-	23	0	20	4	1
le450_15d	450	16750	16	-	23	0	19	4	1
le450_25c	450	17343	26	-	29	0	27	7	2
le450_25d	450	17425	27	-	29	0	27	8	2
flat300_26_0	300	21633	27	-	40	1	33	8	2
flat300_28_0	300	21695	31	-	40	0	34	8	2
flat1000_50_0	1000	245000	92	-	108	2	88	904	185
flat1000_60_0	1000	245830	93	-	108	2	89	917	187
flat1000_76_0	1000	246708	88	-	109	2	89	943	192

4.5 Positionnement de l'algorithme proposé dans un contexte général

Dans cette section, l'algorithme NoN est comparé avec les meilleurs résultats de l'état de l'art et avec les algorithmes mentionnés plus haut afin de mieux apprécier sa contribution. Nous consignons donc dans le tableau 4.4 les résultats obtenus, pour les algorithmes TABUCOL, DSATUR et NoN. Les quatre premières colonnes donnent les caractéristiques du graphe : le nom, le nombre de sommets (n), le nombre des arêtes (m) et le nombre chromatique (χ) lorsqu'il est connu. La colonne suivante indique le meilleur nombre de couleurs connu pour chaque instance et les colonnes 6 – 8 rapportent le nombre de couleurs utilisées par chacun des trois algorithmes. Les caractéristiques des graphes et les données pour TABUCOL (Long_TABU) et DSATUR sont tirées de [30]. Les articles plus récents de Hao et Wu [32], de Malaguti et Toth [6] et de Malaguti

et al. [13] nous ont permis de remplir la colonne 5. Les temps d'exécution avec multithreading (8 threads) en secondes sur chaque instance sont rapportés dans la dernière colonne du tableau.

Tableau 4.4: Résultats obtenus par TABUCOL, DSATUR et NoN sur les graphes DIMACS

Nom	*n*	*m*	X	Meilleur	TABUCOL	DSATUR	NoN	T(NoN)
DSJC125.1	125	736	5	5	5	6	6	0,2
DSJC125.5	125	3891	-	17	17	21	18	0,4
DSJC125.9	125	6961	-	44	44	50	45	0,3
DSJC250.1	250	3218	-	8	8	10	9	0,1
DSJC250.5	250	15668	-	28	29	38	31	1,0
DSJC250.9	250	27897	-	72	72	91	75	3,1
DSJC500.1	500	12458	-	12	13	16	14	0,9
DSJC500.5	500	62624	-	48	50	67	51	13,2
DSJC500.9	500	224874	-	126	130	161	135	42,5
DSJR500.1	500	3555	12	12	12	12	13	0,3
DSJR500.1c	500	121275	85	85	86	87	96	20,2
DSJR500.5	500	58862	122	122	128	130	128	22,3
DSJC1000.1	1000	49629	-	20	22	26	22	11,8
DSJC1000.5	1000	249826	-	83	89	114	90	200,4
DSJC1000.9	1000	449449	-	222	245	297	**239**	730,4
fpsol2.i.1	496	11654	65	65	65	65	65	0,7
fpsol2.i.2	451	8691	30	30	30	30	30	0,4
fpsol2.i.3	425	8688	30	30	30	30	30	0,3
inithx.i.1	864	18707	54	54	54	54	54	2,0
inithx.i.2	645	13979	31	31	31	31	31	0,9
inithx.i.3	621	13969	31	31	31	31	31	0,9
latin_square_10	900	307350	-	97	106	126	108	186,5
le450_15a	450	8168	15	15	15	16	16	0,6
le450_15b	450	8169	15	15	15	16	16	0,5
le450_15c	450	16680	15	15	16	24	20	1,3
le450_15d	450	16750	15	15	16	24	19	1,3
le450_25a	450	8260	25	25	25	25	25	0,7
le450_25b	450	8263	25	25	25	25	25	0,7
le450_25c	450	17343	25	25	27	29	27	1,8
le450_25d	450	17425	25	25	27	28	27	1,8
le450_5a	450	5714	5	5	5	10	5	0,2
le450_5b	450	5734	5	5	5	9	5	0,3
le450_5c	450	9803	5	5	5	6	5	0,3
le450_5d	450	9757	5	5	5	11	5	0,4
mulsol.i.1	197	3925	49	49	49	49	49	0,1
mulsol.i.2	188	3885	31	31	31	31	31	0,1
mulsol.i.3	184	3916	31	31	31	31	31	0,1
mulsol.i.4	185	3946	31	31	31	31	31	0,1
mulsol.i.5	186	3973	31	31	31	31	31	0,1
school1	385	19095	14	14	14	17	14	1,0

(Suite)								
Nom	*n*	*m*	χ	Meilleur	Tabucol	DSATUR	NoN	Temps
school1_nsh	352	14612	14	14	14	25	15	0,7
zeroin.i.1	211	4100	49	49	49	49	49	0,1
zeroin.i.2	211	3541	30	30	30	30	30	0,1
David	87	406	11	11	11	11	11	0,0
Homer	561	1629	13	13	13	13	13	0,2
Huck	74	301	11	11	11	11	11	0,0
Jean	80	254	10	10	10	10	10	0,0
games120	120	638	9	9	9	9	9	0,0
miles1000	128	3216	42	42	42	42	43	0,1
miles1500	128	5198	73	73	73	73	73	0,2
miles250	128	387	8	8	8	8	8	0,0
miles500	128	1170	20	20	20	20	20	0,0
miles750	128	2113	31	31	31	31	31	0,1
queen5_5	25	160	5	5	5	5	5	0,0
queen6_6	36	290	7	7	7	9	8	0,0
queen7_7	49	476	7	7	7	10	9	0,0
queen8_12	96	1368	12	12	12	13	13	0,0
queen8_8	64	728	9	9	9	12	10	0,0
queen9_9	81	2112	10	10	10	14	11	0,0
queen10_10	100	2940	11	11	11	13	12	0,1
queen11_11	121	3960	11	11	11	15	13	0,0
queen12_12	144	5192	12	12	13	15	14	0,1
queen13_13	169	6656	13	14	14	17	15	0,1
queen14_14	196	8372	14	14	15	18	16	0,1
queen15_15	225	10360	15	15	16	19	17	0,2
queen16_16	256	12640	16	16	17	21	18	0,3
myciel3	11	20	4	4	4	4	4	0,0
myciel4	23	71	5	5	5	5	5	0,0
myciel5	47	236	6	6	6	6	6	0,0
myciel6	95	755	7	7	7	7	7	0,0
myciel7	191	2360	8	8	8	8	8	0,0
mugg88_1	88	146	4	4	4	4	4	0,0
mugg88_25	88	146	4	4	4	4	4	0,0
mugg100_1	100	166	4	4	4	4	4	0,0
mugg100_25	100	166	4	4	4	4	4	0,0
abb313GPIA	1557	53356	-	9	11	11	**10**	8,4
ash331GPIA	662	4185	4	4	5	5	**4**	0,4
ash608GPIA	1216	7844	4	4	5	5	5	2,1
ash958GPIA	1916	12506	4	4	6	6	5	7,7
will199GPIA	701	6772	7	7	7	7	7	0,5
1-Insertions_4	67	232	4	4	4	5	5	0,0
1-Insertions_5	202	1227	-	6	6	6	6	0,0
1-Insertions_6	607	6337	-	7	7	7	7	0,3
2-Insertions_3	37	72	4	4	4	4	4	0,0

(Suite)								
Nom	*n*	*m*	χ	Meilleur	Tabucol	DSATUR	NoN	Temps
2-Insertions_4	149	541	4	4	4	5	5	0,0
2-Insertions_5	597	3936	-	6	6	6	6	0,2
3-Insertions_3	56	110	4	4	4	4	4	0,0
3-Insertions_4	281	1046	-	5	5	5	5	0,1
3-Insertions_5	1406	9695	-	6	6	6	6	2,5
4-Insertions_3	79	156	-	4	4	4	4	0,0
4-Insertions_4	475	1795	-	5	5	5	5	0,2
1-FullIns_3	30	100	4	4	4	4	4	0,1
1-FullIns_4	93	593	5	5	5	5	5	0,1
1-FullIns_5	282	3247	6	6	6	6	6	0,2
2-FullIns_3	52	201	5	5	5	5	5	0,0
2-FullIns_4	212	1621	6	6	6	6	6	0,1
2-FullIns_5	852	12201	7	7	7	7	7	1,0
3-FullIns_3	80	346	6	6	6	6	6	0,0
3-FullIns_4	405	3524	7	7	7	7	7	0,2
3-FullIns_5	2030	33571	8	8	8	8	8	10,3
4-FullIns_3	114	541	7	7	7	7	7	0,0
4-FullIns_4	690	6650	8	8	8	8	8	0,4
4-FullIns_5	4146	77305	-	9	9	9	9	75,0
5-FullIns_3	154	792	8	8	8	8	8	0,0
5-FullIns_4	1085	11395	-	9	9	9	9	1,7
wap01	2368	110871	-	42	45	46	45	118,4
wap02	2464	111742	-	41	44	45	44	124,0
wap03	4730	286722	-	44	53	54	**51**	799,5
wap04	5231	294902	-	42	48	48	**46**	917,9
wap05	905	43081	50	50	50	50	50	13,7
wap06	947	43571	40	40	44	46	**43**	14,3
wap07	1809	103368	-	42	45	46	45	79,0
wap08	1870	104176	-	42	45	45	45	82,1
qg.order30	900	26100	30	30	30	30	30	8,4
qg.order40	1600	62400	40	40	40	42	40	47,0
qg.order60	3600	212400	60	60	60	62	60	595,2
qg.order100	10000	990000	100	100	100	103	100	19529,0
flat300_20_0	300	21375	20	20	20	40	20	1,2
flat300_26_0	300	21633	26	26	27	41	33	1,7
flat300_28_0	300	21695	28	28	31	41	34	1,7
flat1000_50_0	1000	245000	50	50	92	112	**88**	185
flat1000_60_0	1000	245830	60	60	93	113	**89**	187
flat1000_76_0	1000	246708	76	82	88	114	**89**	192

En observant le tableau 4.4, on constate qu'à l'exception de trois graphes sur 125 (DSJR500.1, DSJR500.1c et miles1000), l'algorithme NoN est toujours au moins aussi bon si ce n'est meilleur que DSATUR. La différence est plus prononcée à mesure que le nombre de sommets augmente et elle peut dépasser les cinquante couleurs. Par exemple, NoN utilise 239 couleurs pour DSJC1000.9 alors que DSATUR nécessite 297 couleurs, soit une dimunition de 58 couleurs.

Par ailleurs, pour environ 10 graphes, NoN utilise moins de couleurs que TABUCOL comme l'indique les nombres en gras dans le tableau. Ce qui attire particulièrement l'attention, c'est que les graphes concernés sont pour la plupart les graphes réputés « difficiles » comme *les flat1000*, et *les wap*. Lorsque TABUCOL est meilleur que NoN, la différence est souvent inférieure à trois couleurs.

Quand on compare les résultats de NoN aux nombres chromatiques, on se rend compte que cet algorithme donne pour une bonne partie des graphes la coloration optimale et dans le cas contraire, une coloration très proche de l'état de l'art.

L'algorithme NoN est un algorithme constructif dont la performance sur les graphes est non seulement comparable à celle de TABUCOL, mais est parfois meilleure. S'il est vrai que les nombreux travaux de recherche sur les algorithmes hybrides ont permis d'améliorer les résultats des métaheuristiques en général et de TABUCOL en particulier, il est indéniable que ces algorithmes hybrides sont de plus en plus complexes et exigent un long temps de calcul. L'algorithme NoN combine donc la simplicité et la rapidité des algorithmes constructifs et l'efficacité des métaheuristiques. Cet état de chose pourrait s'expliquer par le fait que NoN, quand bien-même sélectionne à chaque fois l'optimum local sans jamais revenir sur sa décision, fonde le choix sur les informations de plusieurs nœuds du graphe.

Toutefois, l'algorithme NoN présente des limites qui sont inhérentes à sa constitution. Étant une heuristique, il n'est pas capable de trouver le nombre chromatique pour tous les graphes. De plus, vu qu'on construit les ensembles C_u pour chaque sommet, certaines opérations sont possiblement répétées durant l'exécution et des améliorations pourraient ainsi être envisagées pour réduire le temps de calcul.

CHAPITRE 5 CONCLUSION ET PERSPECTIVES

La coloration de graphe est un problème classique de la théorie des graphes et a attiré l'attention de nombreux chercheurs en raison de ses multiples applications pratiques et de la complexité de sa résolution. Plusieurs méthodes de résolution ont été suggérées mais aucune n'est capable de résoudre optimalement le problème pour n'importe quel graphe en un temps polynomial. Les méthodes exactes sont inadéquates pour les gros graphes et les algorithmes constructifs, quoique rapides, ne sont pas compétitifs face aux métaheuristiques. En fait, les meilleures performances actuelles sont obtenues par les algorithmes hybrides qui, bien souvent sont très complexes et nécessitent un long temps de calcul.

La principale contribution de ce mémoire est le développement et l'implémentation d'un algorithme constructif efficace dénommé algorithme NoN (**N**eighbours **o**f **N**eighbours) dont les résultats sont comparables à ceux des métaheuristiques. Son principe consiste à colorer en temps polynomial les sommets du graphe, une couleur à la fois, en privilégiant les voisins des voisins des nœuds colorés.

Les résultats des tests effectués sur les graphes du centre DIMACS démontrent que l'algorithme NoN est un algorithme efficace et fiable ; notre objectif de recherche est atteint. Pour un temps d'exécution raisonnable, l'algorithme NoN donne non seulement de meilleurs résultats que les méthodes constructives connues mais ces résultats sont similaires à ceux de TABUCOL, la métaheuristique la plus connue. De même, l'algorithme NoN trouve la coloration optimale pour la majorité des graphes et lorsque ce n'est pas le cas, le nombre de couleurs utilisées n'est jamais loin des meilleurs résultats du moment, tout ceci avec une régularité sans précédent. Il est important de mentionner que l'algorithme NoN est pratiquement insensible à l'ordre de parcours du graphe, point faible des méthodes constructives et son temps d'exécution peut être réduit de façon substantielle avec le multithreading et le parallélisme.

De nombreux travaux peuvent être faits pour rendre l'algorithme NoN beaucoup plus attrayant. Entre autres, réduire le temps d'exécution en ne construisant les ensembles C_u que pour un très petit nombre de sommets u, améliorer l'exactitude des résultats en utilisant une fonction de coloration qui cerne mieux le problème et caractériser les graphes pour lesquels l'algorithme

NoN donne une coloration optimale. Nous avons la ferme conviction que l'algorithme proposé sera le point de départ d'une nouvelle façon d'aborder le problème de la coloration de graphe, avec à l'arrivée une résolution plus efficace d'un des problèmes parmi les plus connus et les plus étudiés des mathématiques discrètes.

BIBLIOGRAPHIE

[1] A. Gamst, "Some lower bounds for a class of frequency assignment problems," *IEEE Transactions on Vehicular Technology*, vol. 35, no. 1, pp. 8-14, 1986.

[2] D. de Werra, "An introduction to timetabling," *European Journal of Operational Research*, vol. 19, no. 2, pp. 151-162, 1985.

[3] D. B. West, *Introduction to graph theory*, vol. 2: Prentice hall Upper Saddle River, NJ, 2001.

[4] Y. M. Chen et W. C. Wang, "An adaptive rescheduling scheme based heuristic algorithm for cloud services applications," *International Conference on Machine Learning and Cybernetics (ICMLC), 2011* vol. 3, 2011, pp. 961-966.

[5] M. R. Gary et D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-completeness," WH Freeman and Company, New York, 1979.

[6] E. Malaguti et P. Toth, "A survey on vertex coloring problems," *International Transactions in Operational Research*, vol. 17, no. 1, pp. 1-34, 2010.

[7] P. Galinier et A. Hertz, "A survey of local search methods for graph coloring," *Computers & Operations Research*, vol. 33, no. 9, pp. 2547-2562, 2006.

[8] J. R. Brown, "Chromatic scheduling and the chromatic number problem," *Management Science*, vol. 19, no. 4-Part-1, pp. 456-463, 1972.

[9] D. Brélaz, "New methods to color the vertices of a graph," *Communications of the ACM*, vol. 22, no. 4, pp. 251-256, 1979.

[10] J. Peemöller, "A correction to Brelaz's modification of Brown's coloring algorithm," *Communications of the ACM*, vol. 26, no. 8, pp. 595-597, 1983.

[11] A. Mehrotra et M. A. Trick, "A column generation approach for graph coloring," *Informs Journal on Computing*, vol. 8, no. 4, pp. 344-354, 1996.

[12] S. Gualandi et F. Malucelli, "Exact solution of graph coloring problems via constraint programming and column generation," *Informs Journal on Computing*, vol. 24, no. 1, pp. 81-100, 2012.

[13] E. Malaguti, M. Monaci, et P. Toth, "An exact approach for the vertex coloring problem," *Discrete Optimization*, vol. 8, no. 2, pp. 174-190, 2011.

[14] I. Méndez-Díaz et P. Zabala, "A branch-and-cut algorithm for graph coloring," *Discrete Applied Mathematics*, vol. 154, no. 5, pp. 826-847, 2006.

[15] I. Méndez-Díaz et P. Zabala, "A cutting plane algorithm for graph coloring," *Discrete Applied Mathematics*, vol. 156, no. 2, pp. 159-179, 2008.

[16] F. T. Leighton, "A graph coloring algorithm for large scheduling problems," *Journal of Research of the National Bureau of Standards*, vol. 84, no. 6, pp. 489-506, 1979.

[17] B. Bollobás et A. Thomason, "Random graphs of small order," in *Random graphs*, vol. 83, 1985, pp. 47-97.

[18] J. C. Culberson et F. Luo, "Exploring the k-colorable landscape with iterated greedy," *Cliques, coloring, and satisfiability: second DIMACS implementation challenge*, vol. 26, pp. 245-284, 1996.

[19] A. Hertz et D. Werra, "Using tabu search techniques for graph coloring," *Computing*, vol. 39, no. 4, pp. 345-351, 1987.

[20] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & Operations Research*, vol. 13, no. 5, pp. 533-549, 1986.

[21] C. Morgenstern, "Distributed coloration neighborhood search," *Discrete Mathematics and Theoretical Computer Science*, vol. 26, pp. 335-358, 1996.

[22] C. Avanthay, A. Hertz, et N. Zufferey, "A variable neighborhood search for graph coloring," *European Journal of Operational Research*, vol. 151, no. 2, pp. 379-388, 2003.

[23] I. Blöchliger et N. Zufferey, "A graph coloring heuristic using partial solutions and a reactive tabu scheme," *Computers & Operations Research*, vol. 35, no. 3, pp. 960-975, 2008.

[24] A. Hertz, M. Plumettaz, et N. Zufferey, "Variable space search for graph coloring," *Discrete Applied Mathematics*, vol. 156, no. 13, pp. 2551-2560, 2008.

[25] D. S. Johnson, C. R. Aragon, L. A. McGeoch, et C. Schevon, "Optimization by simulated annealing: An experimental evaluation; part I, graph partitioning," *Operations research*, vol. 37, no. 6, pp. 865-892, 1989.

[26] L. Davis, "Order-based genetic algorithms and the graph coloring problem," *Handbook of genetic algorithms*, pp. 72-90, 1991.

[27] C. Fleurent et J. A. Ferland, "Genetic and hybrid algorithms for graph coloring," *Annals of Operations Research*, vol. 63, no. 3, pp. 437-461, 1996.

[28] P. Galinier et J. K. Hao, "Hybrid evolutionary algorithms for graph coloring," *Journal of combinatorial optimization*, vol. 3, no. 4, pp. 379-397, 1999.

[29] E. Malaguti, M. Monaci, et P. Toth, "A metaheuristic approach for the vertex coloring problem," *Informs Journal on Computing*, vol. 20, no. 2, pp. 302-316, 2008.

[30] P. Galinier, A. Hertz, et N. Zufferey, "An adaptive memory algorithm for the coloring problem," *Discrete Applied Mathematics*, vol. 156, no. 2, pp. 267-279, 2008.

[31] Q. Wu et J. K. Hao, "Coloring large graphs based on independent set extraction," *Computers & Operations Research*, vol. 39, no. 2, pp. 283-290, 2012.

[32] J. K. Hao et Q. Wu, "Improving the extraction and expansion method for large graph coloring," *Discrete Applied Mathematics*, 2012.

[33] Seuvitor Research. Comparison of algorithms for graph coloring, [En ligne] code.google.com/p/seuvitor-research/wiki/ColoringAlgorithmsComparison. [Consulté le 05 Mai 2013].