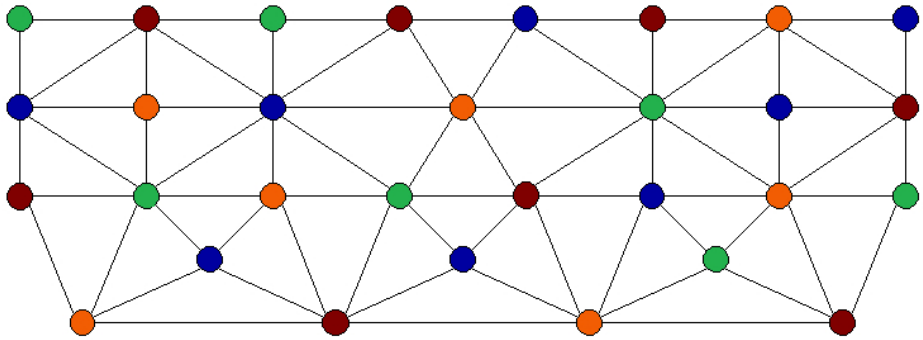
2015-2016

Master I
Informatique

Equitable-Coloring Problem



Rayer Ugo

Sous la direction de M.
Hao Jin-Kao

Soutenu publiquement le :
23 Juin 2016

université angers

REMERCIEMENTS

Pour commencer, je remercie chaleureusement M Jin-Kao Hao pour ce sujet fortement intéressant qui n'a fait que renforcer mon attrait pour le monde de la recherche universitaire. Je tiens ensuite à remercier l'ensemble des camarades avec qui j'ai partagé notre espace de travail pour leur bonne humeur et l'ensemble de nos discussions. Je remercie également M Florian David, pour ces précieux conseils techniques tout au long de ces quelques mois. De plus, merci à Mme Sara Tari pour les différentes discussions sur le monde de l'optimisation combinatoire et ses méthodes qui ont renforcé ma culture du domaine et m'ont aidé dans la réalisation de ce projet. Ensuite, je remercie M David Lesaint pour l'autorisation d'utiliser le cluster du LERIA et Mr Jean-Mathieu Chantrein pour l'assistance technique sur ce dernier. Un grand merci également à ma sœur et éternelle correctrice Josépha.

Table des matières

1. Introduction

Ces pages ont pour vocation de présenter le travail de recherche effectué sous la direction du Pr. Jin-Kao Hao au cours de ma première année de Master Informatique. Ce projet a pour but de valider les connaissances acquises tout au long des enseignements suivis à la Faculté des Sciences de l'Université d'Angers.

Il a pour objet l'étude du problème de coloration équitable au sein d'un graphe. Problème complexe de mathématiques appliquées, les missions qui nous ont été confiées au cours de ce travail d'étude et de recherche furent d'étudier rapidement l'état de la littérature sur ce problème puis d'implémenter une méthode approchée existante pour le résoudre. Enfin, nous devions agrémenter l'algorithme retenu d'idées personnelles et analyser nos résultats expérimentaux.

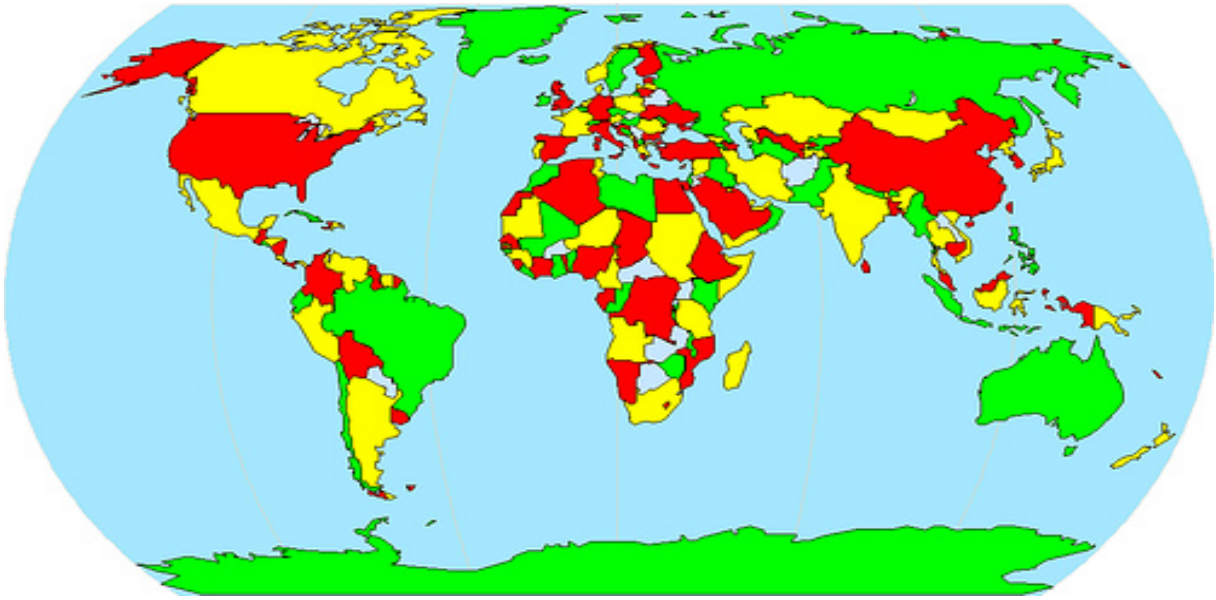
Nous commencerons donc par présenter le problème ainsi que ses applications. Nous ferons ensuite un récapitulatif de la courte phase d'étude de l'existant aussi bien d'un point de vue théorique que des différentes méthodes existantes pour résoudre le problème ECP. Nous détaillerons point par point l'algorithme retenu ainsi que les modifications personnelles apportées. Pour terminer, nous présenterons nos résultats expérimentaux et discuterons rapidement sur ces derniers avant de conclure ce rapport.

2. Présentation du problème

Ce rapport concerne l'étude du problème de coloration équitable au sein d'un graphe. Afin de mieux comprendre ce problème, nous présenterons dans un premier temps la version non-équitable du problème. Dans un second temps, nous spécifierons la version équitable et introduirons certaines de ses applications.

1 Coloration d'un graphe

Considérons un graphe G comme un ensemble de sommets V associé à un ensemble E d'arcs reliant une paire de sommets. Considérons également un stable comme un sous-ensemble S de V tel qu'aucune paire de sommets de S ne soit reliée par un arc dans E. Nous pouvons ainsi définir une coloration de ce graphe G comme une partition des sommets de V en k stables (V1, … , Vk). Chaque stable est également appelé « classe de couleur ».



Problème des 4-couleurs

Un des problèmes les plus connus associés à ce type d'étude est la recherche du nombre chromatique d'un graphe, consistant à déterminer le plus petit nombre de couleurs nécessaires à la coloration de ce graphe. De classe NP-Complet (Holyer [1981]), ce problème, tout comme le nôtre, est difficile à résoudre dans le cas général.

2 Coloration équitable

A l'instar du problème du nombre chromatique, le problème de la coloration équitable est également une recherche du nombre minimal de couleur nécessaire à la coloration d'un graphe. Cependant, une contrainte essentielle est rajoutée : une coloration n'est équitable que si ses classes de couleurs ne diffèrent en termes de nombre d'éléments qu'au plus de 1.

Ce problème trouve plusieurs applications. La première, introduite par W. Meyer en 1973, concerne la planification uniformément répartie sur six jours de différentes tâches [1]. La deuxième application majeure se trouve dans le domaine des probabilités et est issue des travaux de Hajnal et Szemerédi ; elle permet de limiter la variance de sommes de variables aléatoires. Enfin, nous pouvons également citer l'exemple présenté par Tucker où ses recherches se trouvent appliquées dans l'optimisation urbaine de la collecte d'ordures [5].

3. Etude de l'existant

La première phase de ce projet fut d'effectuer une recherche bibliographique sur ce problème encore relativement peu étudié. Ainsi, une semaine a été accordée à cette tâche ce qui m'a permis de mieux appréhender le sujet.

Etant NP-complet, il n'existe à l'heure actuelle aucune méthode exacte de résolution pour un graphe quelconque. Cependant, différentes recherches ont apporté des résultats intéressants sur certaines familles de graphes.

1 Preuves et théorème

N'ayant fait l'objet que de peu d'études théoriques, ce problème n'est concerné que par un seul théorème. Démontré initialement par Hajnal et Szemerédi en 1970, ce théorème éponyme apporte une preuve à la conjecture d'Erdös émise en 1964 :

Théorème 1 : Tout graphe de degré maximum r dispose d'une (r+1)-coloration équitable.

La preuve originale étant relativement longue, plusieurs travaux ont apportés des preuves plus courtes à cette conjecture. Kierstead et Kostochka sont les premiers à y être parvenus en 2008 et présentent leur preuve dans l'article [2] sur lequel nous reviendrons. Différents algorithmes basés sur ce théorème ont été implémentés.

2 Méthodes exactes

Les différents algorithmes présentant une méthode exacte que j'ai pu consulter proposent diverses manières de trouver une (r+1)-coloration équitable. Le premier étudié fut celui présenté dans l'article [2]. L'algorithme proposé par Kierstead et Kostochka est basé sur la notion de *nearly equitable coloration*, coloration où chaque classe de couleur possède la même taille *s* sauf deux : une plus petit contenant $s - 1$ éléments et une plus grande à $s + 1$ sommets.

On peut également citer un des algorithmes de type *Branch-and-Cut* proposé par Bahiense, Frota, Noronha et Ribeiro dans l'article [3] utilisant un modèle en programmation linéaire en nombres entiers. Pour terminer, citons également l'algorithme ECOPT décrit dans la thèse de Daniel E. Severin, utilisant également une mécanique *Branch-and-Cut* et divers composants (heuristiques greedy et DSATUR, recherche tabou, ...) [4].

Toutefois il est intéressant de noter que $r+1$ est rarement le nombre minimal de couleur nécessaire à une répartition équitable et que les méthodes exactes ne sont applicables qu'à des graphes d'ordre relativement faible.

3 Méthodes approchées

A ce jour de nombreux algorithmes de différents types ont été mis au point afin de minimiser le nombre de couleurs nécessaires à une k-coloration équitable. On retrouve notamment des algorithmes basés sur la saturation de degrés (DSATUR) [6] ou encore modélisé par représentation linéaire en nombres entiers [7]. Enfin, différentes recherches locales et algorithmes génétiques ont également été créées. Nous étudierons en détail l'article [8] présentant une recherche tabou itérée avec backtracking pour résoudre notre problème.

4. Présentation de BITS

L'algorithme que nous avons décidé d'implémenter pour ce travail d'étude et de recherche m'a été proposé par mon référent. Il est présenté dans l'article *Backtracking Based Iterated Tabu Search for Equitable Coloring* par Xiangjing Lai, Jin-Kao Hao et Fred Glover [0]. Le principe de cet algorithme est d'utiliser de manière récursive un nombre de moins en moins important de couleurs au sein d'une heuristique tabou visant à trouver une solution libre de tout conflit. Chaque point essentiel de l'algorithme sera brièvement décrit ainsi que certains points-clés de l'implémentation que nous en avons fait.

1 Mécanisme global

La procédure générale de l'algorithme est simple. Elle prend en argument le graphe G étudié, la profondeur η de backtracking, la profondeur α de la recherche tabou et le nombre β de perturbations autorisées et renvoie la coloration équitable avec le plus petit nombre de couleurs trouvée durant le temps imparti.

En premier lieu, nous utilisons une recherche binaire afin de déterminer une valeur initiale k_{best} ainsi qu'une k_{best}-coloration correspondante. Ensuite, notre algorithme va itérativement essayer de réduire ce nombre de couleur tant qu'il reste du temps alloué à l'exécution.

Réduire ce nombre revient à trouver une coloration équitable avec $k' = (k_{best} - n)$ couleurs. Pour tenter d'y parvenir, nous utilisons une recherche tabou itérée avec k' couleurs, n variant de 1 à η. Dans le cas où aucune amélioration n'est apportée depuis η itérations ou si $k' = 2$, on repositionne k' à $k_{best} - 1$, sinon on décrémente simplement k'.

Algorithme général

```
( k^r, s^r) ← Recherche Binaire(G, ?)
k* ←  k^r , k ← k*, s* ←  s^r
Faire
        Si k = k* - η ou k = 2 Alors
                k ← k* - 1
        Sinon
                k ← k - 1
        FinSi
        s ← Recherche tabou itérée( k, G, ?, β)
        Si Evalutation(s) = 0 Alors
                k* ← k, s* ← s
        FinSi
Jusqu'à Timer() > TpsAlloué
Retourner (k*, s*)
```

La fonction d'évaluation utilisée tout au long de l'algorithme dénombre les sommets en conflits, un sommet étant conflictuel s'il appartient à la même classe de couleur qu'un de ses sommets adjacents. Nous tentons donc de minimiser le nombre *k* de couleurs utilisées par le biais d'une recherche tabou itérée au sein de laquelle nous nous déplaçons dans l'espace de recherche du *k*-ECP problème correspondant en minimisant cette fonction d'évaluation.

En terme d'implémentation, cette mécanique ne requiert pas de commentaires particuliers car elle a été très simple à mettre en place.

2 Initialisation

Comme dans tout algorithme de recherche locale, celui ci dispose d'une procédure spécifique d'initialisation. Pour ce faire, nous allions un algorithme de type greedy à une procédure de recherche binaire permettant de déterminer une valeur initiale de *k* acceptable.

Cette procédure est basée sur l'utilisation de deux variables *Uk* et *Lk* représentant les bornes supérieure et inférieure de *k.* Elles sont utilisées au sein d'une recherche tabou suivant le même schéma que notre recherche principale. Cependant, la profondeur de recherche est faible, ceci dans le but d'intensifier la recherche.

L'algorithme implémenté est le suivant et reçoit en paramètre le graphe G à étudier :

Algorithme 1 : Recherche Binaire

```
Uk ←|V|, Lk ← 0
TantQue Uk > Lk + 1
    k ← floor( (Uk + Lk) / 2 )
    s ← Initialisation(G, k)
    s ← Tabu_Search(s, 100)
    Si Evaluation(s) = 0 Alors
        sr ← s
        kr ← k
        Uk ← k
    Sinon
        Lk ← k
    FinSi
FinTantQue
Retourner (kr, sr)
```

L'algorithme greedy d'initialisation reçoit quant à lui le graphe et un nombre de couleurs pour la répartition à effectuer et renvoie la coloration obtenue via le pseudo-code suivant :

Algorithme 2 : Initialisation

```
Pour i de 0 à (k-1)
    Vk[i] ← ?
FinPour
U ← V /* U est l'ensemble des sommets non assignés à une couleur */
Pour i de 0 à (k-1)
    Sélectionner un sommet quelconque v € U
    Vk[i] ← Vk[i] + {v}
    U ← U \ {v}
    Mise à jour de M
FinPour
i ← 0
TantQue U != ?
    v ← argmin( |Ni(v')| : v' € U ) /* Ni est l'ensemble des adjacents de v dans Vk[i] */
    Vk[i] ← Vk[i] + {v}
```

```
        U ← U \ {v}
        Mise à jour de M
        i ← (i+1) % k
    FinTantQue
```

En terme d'implémentation, les différentes classes de couleurs sont contenues dans des vecteurs d'entiers et nous profitons de cette procédure pour initialiser la matrice d'adjacence M que nous présenterons plus tard.

Enfin, il est important de noter que nous n'avons pas besoin de briser les égalités de manière aléatoire comme cela est présenté dans l'article. En effet, nos sommets étant également stockés dans un vecteur, nous mélangeons aléatoirement celui-ci via la méthode *shuffle* offerte par la librairie *algorithm* au début de la procédure. Ainsi, l'aléatoire agit de manière efficace sans être trop coûteuse (linéaire sur le nombre d'éléments -1).

3 Recherche tabou itérée

La procédure de recherche utilisée dans notre fonction principale est une recherche itérée qui prend en paramètre le graphe G étudié, le nombre initial de couleur *k*, le paramètre β et la profondeur α de la recherche tabou utilisée au coeur de celle-ci. En sortie, cette recherche itérée fournit la meilleure solution trouvée par l'algorithme suivant :

Algorithme 3 : Recherche tabou itérée

```
Solution s ← Initialisation(G,k)
s ← Recherche_tabou(s, ?)
d ← 0 /* compte le nombre de perturbations sans mise à jour */
Faire
        Solution s' ← Random_Walk(s)
        s' ← Recherche_tabou(s', ?)
        Si Evaluation(s') < Evaluation(s) Alors
                s ← s'
                d ← 0
        Sinon
                d ← d +1
        FinSi
Jusqu'à d = β ou Evaluation(s) = 0
Retourner s
```

Comme nous pouvons le voir, cette procédure initialise une solution via une première recherche tabou basée sur une coloration issue de notre algorithme *greedy*. Ensuite, nous perturbons cette solution courante avant de tenter de l'améliorer via une autre recherche tabou. Nous ne nous arrêtons que lorsqu'on rencontre une coloration équitable sans conflit ou si la meilleure solution n'a pas été mise à jour après β itérations. Au niveau de l'implémentation, nous avons rajouté la gestion du temps restant au sein de cette procédure à la fois dans le critère d'arrêt mais également dans la recherche tabou employée que nous allons maintenant présenter avant de revenir sur la fonction de perturbation.

4 Recherche tabou

Fonctionnant de manière classique, cette procédure reçoit en entrée une coloration initiale s_0, une fonction de voisinage N et une profondeur de recherche α et retourne la meilleure coloration trouvée durant le processus.

Le principe de cette recherche est relativement simple. A chaque itération, le meilleur voisin admissible remplace la solution courante. Selon les différents cas, un ou plusieurs mouvements sont alors ajoutés à la liste tabou et ce processus est répété tant qu'au moins un des critères d'arrêt n'est pas atteint. Ils sont ici au nombre de deux.

Le premier est l'obtention d'une coloration sans conflit. Le second dépend de la profondeur α passée en paramètre. En effet, dans le cas où la procédure n'a pas réussi à améliorer la meilleure solution pendant α itérations, celle-ci s'arrête. Il est important de noter qu'un voisin est admissible s'il n'est pas interdit par la liste tabou ou si son évaluation est meilleure que celle de la coloration optimale trouvée jusqu'alors.

Algorithme 4 : Recherche tabou

```
Solution s ← s0
Solution sb ← s
d ← 0 /* compte le nombre d'itérations sans mise à jour de sb */
Faire
        Solution s' ← BestAdmissibleNeighboor(s)
        s ← s'
        Mise à jour de la liste tabou
        Si Evaluation(s) < Evaluation(sb) Alors
                sb ← s
                d ← 0
        Sinon
                d ← d +1
        FinSi
Jusqu'à d =  ? ou Evaluation(s) = 0
Retourner  sb
```

4.1. Voisinage

La relation de voisinage définie dans **BITS** est en réalité l'union de deux voisinages distincts et bornés de manière raisonnable. Ces deux ensembles sont définis à partir de l'ensemble des sommets en conflit dans une solution *s* donnée. Un sommet est en conflit s'il appartient à la même classe de couleurs qu'au moins un de ses sommets adjacents.

Soit C(*s*) cet ensemble de sommets. Le premier opérateur de voisinage, *OneMove*, se définit comme le déplacement d'un sommet *v* en conflit appartenant à la classe de couleurs *i* vers une autre classe de couleurs *j*, tout en respectant les contraintes de répartition (i.e. le sommet *v* appartient à une classe de couleurs *i* ayant un nombre d'éléments correspondant à la limite supérieure).

– $N_1(s)$: { (*v, i, j*) | *v* € C(*s*), *v*€Vk_i, i != j, $|Vk_i|$ > floor(|V| /k), $|Vk_j|$ < top(|V| / k) }

Le second voisinage exploré par l'algorithme correspond à l'application de l'opérateur *Swap* à une solution *s*. Cet opérateur définit des couples de sommets dont au moins l'un des deux est en conflit et appartenant à deux partitions différentes.

– $N_2(s)$: { (*u,v*) | *u* € Vk_i , *v* € Vk_j , i != j, {*u,v*} ∩ C(*s*) != ∅ }

4.2. Sélection du meilleur voisin admissible

La mécanique de cette recherche locale étant tabou, nous sélectionnons à chaque itération le meilleur voisin non interdit par la liste tabou, exception faite du cas où son évaluation est meilleure que l'évaluation de la meilleure solution trouvée jusqu'alors (mécanisme d'aspiration). Ainsi, il a été nécessaire de mettre en place un système d'évaluation incrémentale permettant d'observer rapidement le gain associé à chaque voisin. De fait, nous maintenons tout au long de la durée de vie d'une solution une matrice d'adjacence M de |V|*k qui dénombre pour chaque sommet le nombre de sommets adjacents appartenant à chacune des classes de couleurs.

L'évaluation incrémentale d'un voisin $N_1(s)=(v, i, j)$ se fait donc par le biais de la matrice M de la manière suivante :

- $\Delta((v,i,j)) = M[v][j] - M[v][i]$

La matrice M peut ensuite être mise à jour de la manière suivante. Pour chaque sommet u adjacent à v, $M[u][i] \leftarrow M[u][i] - 1$, $M[u][j] \leftarrow M[u][j] + 1$.

Pour un voisin $N_2(s)=(u,v)$ l'évaluation se calcule de la manière suivante :

- $\Delta((u,v)) = (M[v][k(u)] - M[v][k(v)]) + (M[u][k(v)] - M[u][k(u)]) - 2*\delta(u,v)$ où $\delta(u,v) = 1$ si (u,v) € E, 0 sinon.

Nous pouvons noter que la mise à jour de M pour un voisin de type *Swap* peut se faire via la même fonction que pour un voisin *OneMove* mais appliquée deux fois. En effet, nous pouvons voir qu'un *Swap* (u,v) correspond à deux *OneMove* $(u,k(u),k(v))$, $(v,k(v),k(u))$ consécutifs.

Le voisinage d'une solution est donc l'union de N_1 et N_2 et qu'il est borné par O(C(s)*(k+ |V|)). Au niveau de l'implémentation, la structure de données employées n'est actuellement pas optimisée mais fonctionnelle. En effet, la gestion n'en est pas dynamique et le voisinage d'une solution est calculé dès qu'il est nécessaire. Par exemple, l'utilisation de l'outil *callgrind* pendant l'exécution de *valgrind* sur notre programme nous indique que le calcul du voisinage *Swap* représente environ 40% du temps d'exécution total.

4.3. Gestion de la liste tabou

Le dernier point-clé de cette recherche locale est la liste tabou. Celle-ci a pour but d'interdire à un (ou plusieurs) mouvement(s) effectué(s) d'être répété(s) avant un certain nombre d'itérations. Le délai entre deux applications d'un même mouvement est donc soumis à un paramètre appelé par la suite *tabu tenure (*i.e. *tt*). Ainsi, une fois qu'un voisin (*v, i, j*) de type *OneMove* a été choisi, il convient d'interdire au sommet *v* de retourner dans la classe de couleurs *i* avant *tt* itérations, exception faite s'il améliore la meilleure solution trouvée jusqu'alors. De la même manière, lorsque le voisin est de type *Swap,* nous interdisons à la fois au sommet *u* et au sommet *v* de retourner dans leur ancienne classe de couleurs.

4.3.1. Matrice tabou

Dans cette optique, une nouvelle matrice MT, également implémentée via un vecteur de vecteurs, de taille |V| * *k* et initialisée à 0 est gérée dynamiquement tout au long de la recherche. A chaque itération *x*, nous mettons à jour les cases concernées de la manière suivante :

- *OneMove* (*v, i, j*) : MT[*v*][*i*] = *x* + *tt*
- *Swap (u,v)* : MT[*v*][*k(v)*] = *x* + *tt* , MT[*u*][*k(u)*] = *x* + *tt*

De ce fait, un voisin de type *Swap* est représenté par 4 entiers afin d'avoir constamment accès facilement aux classes de couleurs concernées.

4.3.2. Tabu tenure

Enfin, le dernier point à éclaircir est la *tabu tenure* précédemment citée. Elle est ici mise à jour dynamiquement en fonction de l'itération *x* et suit alternativement trois règles différentes :

- Règle 1 : *tt(x)* = C_0 + rand(C_1) où C_0 et C_1 sont deux nombres constants empiriquement fixés à 5.
- Règle 2 :*tt(x)* = α * |C(s)| + rand(β) où α et β sont deux paramètres fixés à 0,9 et 5, introduite par Galinier P. et Hao J-K mais avec des valeurs différentes que celles retenues par l'expérimentation [8].
- La règle 3 est plus particulière et ajuste *tt* via une fonction périodique. Pour chaque période, la *tabu tenure* est définie par une séquence de valeurs (a_1, …, a_{p+1}) et une séquence de marges d'intervalle (x_1, …, x_{p+1}) telles que pour chaque itération x dans [x_i, x_{i+1} -1], tt(x) = a_i + rand(2). Ici p est fixé à 15, $(a)_{i = 1...15}$ = (T_{max} / 8) * [1,2,1,4,1,2,1,8,1,2,1,4,1,2,1] où T_{max} est un paramètre représentant la valeur

maximale de ce paramètre et ici fixé à 80. Les marges d'intervalle sont définies par $x_1 = 1$, $x_{i+1} = x_i + 3a_i$, $i \leq 15$.

Ces trois règles sont appliquées à tour de rôle pendant *y (* ici fixé à 3*10^4) itérations.

5 Perturbations d'une solution

Pour terminer la présentation de cet algorithme, il convient d'introduire les deux fonctions de perturbation utilisées au cours de la recherche itérée. L'une suit une mécanique aléatoire tandis que l'autre agit de manière dirigée. Afin de déterminer quelle sera la perturbation effectuée, nous utilisons une variable aléatoire *p* représentant la probabilité que l'aléatoire soit retenue. Cette probabilité est fixée ici de manière empirique à 0,7.

5.1. Perturbation dirigée

Cette procédure simple effectue un nombre η_1 (arbitrairement fixé à 5*10^3) *OneMove* ou *Swap*, choisissant à chaque pas le meilleur admissible et maintient également une liste tabou avec une *tabu tenure* fixée à chaque itération à *tt* = 2000 + rand(1000).

Au niveau de l'implémentation, cette procédure nous a posé énormément de problèmes. Pensant tout d'abord à de difficultés liées à notre niveau technique, nous nous sommes finalement rendu compte après de nombreuses phases d'essai d'implémentations différentes que le problème était issu d'une faiblesse dans notre algorithme. En effet, nous n'avions pas considéré le fait qu'une *k*-coloration équitable était potentiellement accessible lors de cette phase de perturbation. Nous avons donc rajouté un mécanisme de contrôle afin de régler ce soucis. Entre temps, un benchmark complet a été effectué en n'effectuant uniquement que des perturbations aléatoires, ce qui nous permettra de mettre en lumière l'apport de cette mécanique dirigée.

5.2. Perturbation aléatoire

Cette procédure effectue itérativement η_2 mouvement de voisinage de type *Swap* choisi aléatoirement parmi l'ensemble des voisins possibles. Toutefois, nous ne calculons pas l'ensemble des voisins possibles mais sélectionnons un couple de sommets (*u*,*v*) tel que le sommet *u* soit en conflit. Le sommet *v* est sélectionné parmi l'ensemble des autres sommets. Dans l'implémentation proposée dans l'article, η_2 est arbitrairement fixé à 0,3*|V|.

5. Modifications personnelles

Durant la découverte de l'algorithme **BITS** nous avons pu observer que différents paramètres étaient fixés de manière empirique. Ne comprenant pas comment pouvaient être justifiés de tels choix, nous avons apporté quelques modifications à la gestion de ces paramètres avec pour objectif principal d'observer si l'ordre du graphe étudié et l'état courant de la recherche pouvaient influer de manière positive sur les résultats initiaux.

1 Variable aléatoire *p*

Pour commencer, nous avons décidé de lier la probabilité d'exécuter une perturbation aléatoire à l'état courant de la recherche. En effet, il nous a paru intéressant de pouvoir constater par les faits l'idée suivante : la probabilité d'une perturbation aléatoire sera d'autant plus forte que la meilleure solution trouvée n'a pas été améliorée depuis longtemps.

De fait, nous introduisons un nouveau paramètre p_{max} correspondant à la probabilité maximale d'une perturbation aléatoire. Nous fixons dans un premier temps ce paramètre à 0,8 et ajoutons le système de variation suivant : dans le cas de base où la solution vient d'être améliorée, les perturbations sont équiprobables. Ensuite, la probabilité de la perturbation aléatoire augmente en fonction du nombre d'itérations sans amélioration jusqu'à atteindre p_{max} dans le dernier cas.

– Soit *d* itérations depuis la dernière itérations et β le nombre maximal d'itérations :

$$p = 0,5 + (d+1)*(p_{max} - 0,5) / \beta$$

2 Nombre de perturbations η_1 et η_2

Les deux paramètres η_1 et η_2 sont également fixés empiriquement. Bien que η_2 soit dépendant de l'ordre du graphe, η_1 est totalement arbitraire et ne dépend d'aucun invariant du programme.

Étudions dans un premier temps η_1 : est-il logique, bien qu'il s'agisse d'une perturbation dirigée, qu'on effectue autant de mouvements pour perturber une solution de 125 sommets que pour une solution de 1000 ? A nos yeux, non, c'est pourquoi, plutôt que d'effectuer 5000 *Swaps*, nous avons décidé de lier ce paramètre au graphe étudié. Ainsi, η_1 sera égal à $\delta * |V|$ où

δ, que l'on nommera coefficient de perturbation dirigée, sera initialement fixé à 5 et pourra être soumis à une étude ultérieure.

Voyons à présent comment η_2 est géré dynamiquement. En considérant qu'une perturbation fait potentiellement suite à une recherche infructueuse, nous avons voulu observer l'influence que pouvait avoir le paramètre *d* précédemment cité sur η_2. De ce fait, nous avons légèrement augmenté la valeur maximale de $\eta_2 = \lambda * |V|$ en augmentant la valeur maximale de λ à 0,4. Cependant, nous introduisons une borne inférieure correspondant à $\lambda = 0,2$, valeur prise lors d'une perturbation aléatoire avec $d = 0$.

– Soit d itérations depuis la dernière amélioration, λ_{min} et λ_{max} les bornes inférieures et supérieure de λ et β le nombre maximal d'itérations sans amélioration :

$$\eta_2 = \lambda_{min} + (\lambda_{max} - \lambda_{min})*(d + 1) / \beta * |V|$$

3 Application des règles de *tabu tenure*

Pour terminer, nous avons décidé d'apporter une légère modification à l'application des règles de gestion de la *tabu tenure*. Jusqu'alors appliquées pendant *y = 3*10^4 itérations, nous avons préféré réduire la taille de ces périodes afin de pouvoir effectuer au cours d'une recherche itérée plusieurs applications de chacune des règles. Ainsi, nous appliquerons successivement chacune des règles pendant α / 8 itérations, α étant la profondeur de la recherche tabou utilisée.

6. Résultats expérimentaux

Nous présentons ici les conditions d'expérimentation et les différents paramètres du programme ainsi que les instances testées. Nous comparons ensuite les résultats de l'implémentation originale avec la nôtre. Enfin, nous présentons l'influence de l'ensemble de nos modifications ainsi que leur impact unitaire sur les résultats obtenus.

Les 15 instances testées font toutes partie de la littérature de la théorie des graphes et sont téléchargeables ici http://www.info.univ-angers.fr/pub/porumbel/graphs/. Pour chacun de nos benchmarks, chaque instance a été testées au minimum 5 fois sur une période de 3600 secondes.

Les deux premiers benchmark ont été effectués sur notre ordinateur personnel. Le code source a été compilé avec G++ 4.8.4 et les options de compilation -Wall et -O2. Menés à bien

avec un processeur Intel Core I5 M560 (2,67GHz 4Gb RAM), ils ont pour but de comparer les résultats de l'article de référence avec notre implémentation (perturbations aléatoires uniquement et perturbations mixtes). Nous avons également profité du cluster généreusement mis à notre disposition pour effectuer quatre benchmark sur les modifications personnelles que nous avons apportées à **BITS**. Ainsi, nous comparons les résultats de chacune des modifications isolément et conjointement. Ces benchmarks ont été menés à bien sur des processeurs Intel Xeon E5-2670 (2,5GHz 4Gb RAM). Le code source a été recompilé sur le cluster avec G++ 4.8.2 et les options -Wall, -O2 et -static-libstdc++.

Ci-dessous un récapitulatif des paramètres importants et leurs valeurs dans **BITS** et dans notre version modifiée :

Paramètre	Description	Valeur BITS	Valeur perso
η	Profondeur du backtracking	4	4
β	Nombre max d'itérations sans améliorations	30	30
α	Profondeur de la recherche tabou	Binaire : 10^2 Classique : 10^5	Binaire : 10^2 Classique : 10^5
Tmax	Maximum *tabu tenure*	80	80
y	Nompre d'applications de chaque règle	$3*10^4$	Dynamique
η2	Nombre de perturbations aléatoires	0,3*|V|	Dynamique
η1	Nombre de perturbations dirigées	$5*10^3$	Dynamique
p	Probabilité de perturbation aléatoire	0,7	Dynamique

Tableau 1 : Paramètres importants dans BITS

Nous retrouvons dans le tableau suivant pour chaque instance son nom, le nombre de sommets du graphe, le meilleur résultat **K*** trouvé par l'algorithme **BITS,** notre meilleur résultat **Kbest**, le résultat moyen **Kavg**, le nombre d'exécutions **NbExe** et le taux d'obtention de **Kbest** en dernière colonne. En rouge apparaissent les résultats moins bon lorsque la perturbation est mixte. Ceux améliorant sont eux affichés en gras.

			Perturbation aléatoire uniquement				Perturbation mixte			
Instance	**| V |**	**K***	**Kbest**	**Kavg**	**NbExe**	**SR**	**Kbest**	**Kavg**	**NbExe**	**SR**
DSJC125.5	125	17	18	18,6	5	2 / 5	18	**18,2**	5	4 / 5
DSJC250.5	250	30	32	32	5	5 / 5	32	32	5	5 / 5
DSJC250.9	250	72	73	73	5	5 / 5	73	73	5	5 / 5
DSJC500.1	500	13	13	13	5	5 / 5	13	13	13	6 / 6
DSJC500.5	500	56	57	58.4	5	3 / 5	57	**58,2**	5	4 / 5
DSJC500.9	500	129	133	133,2	5	4 / 5	133	133,4	5	3 / 5
DSJR500.5	500	126	127	127,67	6	2 / 6	128	128	5	5 / 5
DSJC1000.1	1000	21	22	22	5	5 / 5	22	22	5	5 / 5
DSJC1000.5	1000	103	112	112	5	5 / 5	112	112	5	5 / 5
DSJC1000.9	1000	252	264	264,2	5	4 / 5	264	265,6	5	2 / 5
R125.1	125	5	5	5	5	5 / 5	5	5	5	5 / 5
R125.5	125	36	36	36	5	5 / 5	36	36	5	5 / 5
R250.5	250	66	66	66.86	7	1 / 7	67	67	5	5 / 5
R1000.5	1000	250	258	258,6	5	2 / 5	259	259,2	5	1 / 5
latin_square	900	115	130	130,8	5	3 / 5	130	131,6	5	2 / 5
Flat300_28	300	34	35	35,8	5	3 / 5	35	**35,6**	5	4 / 5
Flat1000_76	1000	102	112	112	5	5 / 5	112	112	5	5 / 5

Tableau 2: Résultats expérimentaux de BITS

Enfin, le dernier tableau met en lumière l'impact apporté par nos modifications aux résultats obtenus avec notre implémentation et les paramètres initiaux de **BITS**. On peut également voir l'influence unitaire de chaque modification. Ainsi, **ECP2** désigne les résultats avec toutes les modifications actives, **ECP_Tt** avec la *tabu tenure* uniquement, **ECP_Rp** avec la gestion dynamique de la variable aléatoire *p* et enfin **ECP_Nbp** avec le nombre de perturbations effectuées lié à l'ordre du graphe étudié. Pour chacun de ces benchmars, nous présentons les valeurs **Kbest**, **Kavg** et **SR** suivant les mêmes définitions que précédemment.
Nous faisons également un rappel de la valeur optimale **K*** issue de l'article et des valeurs **K_opt** et **Kavg_opt** correspondant aux meilleurs valeurs prises par notre implémentation (i.e. issues du tableau 1). Les résultats figurant en gras sont ceux améliorant les valeurs **K_opt** ou **Kavg_opt**, ceux en rouges apportant la meilleure améliorations.

					ECP2			ECP_Nbp			ECP_Rp			ECP_Tt		
Instance	**| V |**	**K***	**K_opt**	**Kavg_opt**	**Kbest**	**Kavg**	**SR**	**Kbest**	**Kavg**	**SR**	**Kbest**	**Kavg**	**SR**	**Kbest**	**Kavg**	**SR**
DSJC125.5	125	17	18	18,2	**17**	18,2	1 / 5	**17**	18,2	1 / 5	18	18,2	4 / 5	18	18,4	3 / 5
DSJC250.5	250	30	32	32	32	32	5 / 5	32	32	5 / 5	32	32	5 / 5	32	32	5 / 5
DSJC250.9	250	72	73	73	**72**	**72,8**	1 / 5	73	73	5 / 5	**72**	**72,6**	2 / 5	**72**	**72,8**	1 / 5
DSJC500.1	500	13	13	13	13	13	5 / 5	13	13	5 / 5	13	13	5 / 5	13	13	5 / 5
DSJC500.5	500	56	57	58,2	57	58,2	1 / 5	57	**57,2**	4 / 5	57	**57**	5 / 5	57	**57,4**	3 / 5
DSJC500.9	500	129	133	133,2	**131**	**131,8**	1 / 5	133	133,2	4 / 5	**132**	133,2	1 / 5	**131**	**131,8**	1 / 5
DSJR500.5	500	126	127	127,67	127	**127,4**	2 / 5	128	128	5 / 5	128	128	5 / 5	127	**127,6**	2 / 5
DSJC1000.1	1000	21	22	22	22	22	5 / 5	22	22	5 / 5	22	22	5 / 5	22	22	5 / 5
DSJC1000.5	1000	103	112	112	112	112	5 / 5	112	112	5 / 5	112	112	5 / 5	112	112	5 / 5
DSJC1000.9	1000	252	258	258,6	**257**	259,4	1 / 5	**257**	259,2	1 / 5	258	259,4	5 / 5	258	260	1 / 5
R250.5	250	66	66	66,86	67	67	5 / 5	66	**66,8**	1 / 5	67	67	5 / 5	66	**66,8**	1 / 5
R1000.5	1000	250	258	258,6	**257**	**257,8**	1 / 5	**256**	**257**	1 / 5	**256**	**257,6**	1 / 5	**255**	**256,2**	1 / 5
latin_square	900	115	130	130,8	**129**	**129,8**	1 / 5	130	**130**	5 / 5	130	**130**	5 / 5	130	**130**	5 / 5
Flat300_28	300	34	35	35,6	35	**35**	5 / 5	35	**35**	5 / 5	35	**35**	5 / 5	35	35,6	3 / 5
Flat1000_76	1000	102	112	112	112	112	5 / 5	112	112	5 / 5	112	112	5 / 5	112	112	5 / 5

Tableau 3 : Résultats expérimentaux de nos modifications personnelles

7. Analyse et discussions

Nous constatons après lecture complète des résultats que notre implémentation (sans modifications) parvient à égaler pour trois instances la valeur optimale trouvée par l'implémentation originale lorsque la perturbation est uniquement aléatoire et une valeur approchée à 1 pour 6 autres instances. Dans les autres cas, les résultats obtenus sont un peu plus éloignés mais jamais à plus de 9%.

Nous observons ensuite que lorsque la perturbation est mixte les résultats ne sont pas globalement meilleurs. En effet, les valeurs optimales ne sont jamais meilleures, les valeurs moyennes améliorées dans trois cas et détériorées dans cinq. L'absence d'influence majeure de la perturbation dirigée peut s'expliquer par la relative faiblesse de la structure de données utilisée pour représenter le voisinage d'une solution. En effet, celle-ci est également utilisée lors de la perturbation dirigée ; elle est donc très probablement à l'origine de ce manque d'améliorations.

Concernant nos modifications personnelles, nous pouvons observer qu'elles ont une influence non-négligeable. En effet, elles améliorent dans un tiers des cas le nombre de couleurs nécessaires et obtiennent la valeur **K*** dans deux cas supplémentaires. De plus, la valeur moyenne **Kavg** est améliorée pour cinquante pour cent des instances testées. Individuellement, les modifications impactent les résultats dans des proportions équivalentes (huit améliorations sur **Kbest**/**Kavg**) mais sur différentes instances (dix sont impactées au total mais pas par chacune des modifications).

Pour terminer, il nous semblerait intéressant de pouvoir observer l'effet des modifications apportées sur l'implémentation originale de cet algorithme. En effet, malgré la faiblesse de notre perturbation dirigée, les modifications améliorent tout de même significativement nos résultats ce qui nous amène à penser qu'elles pourraient également améliorer ceux de références.

8.Conclusion

L'objet principal de ce projet était d'étudier le problème de coloration équitable au sein d'un graphe. De ce point de vue, nous pouvons considérer ce projet comme une réussite puisque nous avons pu à la fois réaliser une étude documentaire du sujet ainsi que la mise en œuvre d'une méthode de résolution approchée.

Cependant, nous pouvons regretter de ne pas avoir réalisé ce projet à plusieurs. En effet, le temps consacré à l'implémentation de l'algorithme retenu fut très important. Il aurait pu être réduit si ce projet avait été réalisé en binôme, ce qui nous aurait permis d'explorer de

nouvelles pistes pour les modifications envisagées. Toutefois, les résultats obtenus sont à nos yeux satisfaisants.

Pour conclure, nous remercions encore une fois M. Jin-Kao Hao pour ce sujet très intéressant qui nous a permis d'augmenter significativement nos connaissances dans le domaine de l'optimisation combinatoire mais également notre niveau technique.

9.Bibliographie

[0] Xiangjing Lai, Jin-Kao Hao, Fred Glover : « Backtracking Based Iterated Tabu Search for Equitable Coloring. » *Engineering Applications of Artificial Intelligence 46: 269-278, 2015*

[1] W.Meyer : « Equitable Coloring. » *American Mathematical Monthly 80(1973)143–1*

[2] H. A. Kierstead, A. V. Kostochka, M. Mydlarz, E. Szemeredi : « A fast algorithm for equitable coloring. » Combinatorica 30 (2) (2010) 217–224

[3] L. Bahiense , Y. Frota , T. F. Noronha , C. C. Ribeiro : « A Branch-and-Cut Algorithm for the Equitable Coloring Problem Using a Formulation by Representatives. » Article publique de l'Université Fédérale de Rio de Janeiro

[4] D. E Severin : « Estudio poliedral y algoritmo branch-and-cut para el problema de coloreo equitativo en grafos. » Thèse disponible sur le site de l'Université Nationale de Rosario.

[5] A. Tucker. « Perfect graphs and an application to optimizing municipal services. » SIAM Review

[6] I. Méndez-Díaz, G. Nasini, D. E. Severín : « A DSATUR-based algorithm for the Equitable Coloring Problem. » Computers & Operations Research 12-2014

[7] I. Méndez-Díaz, G. Nasini, D. E. Severín : « A polyhedral approach for the equitable coloring problem. » Discrete Applied Mathematics 12-2012

[8] Galinier P., Hao J-K.: « Hybrid evolutionary algorithms for graph coloring. » J. Comb. Optim. 3(4), 379-397(1999)

[9] Kenneth Sôrensen : « Metaheuristics : the Metaphor Exposed. » 09-2012

ENGAGEMENT DE NON PLAGIAT

Je, soussigné(e) Rayer Ugo
déclare être pleinement conscient(e) que le plagiat de documents ou d'une partie d'un document publiée sur toutes formes de support, y compris l'internet, constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.
En conséquence, je m'engage à citer toutes les sources que j'ai utilisées pour écrire ce rapport ou mémoire.

Le 16 Juin 2016

Cet engagement de non plagiat doit être signé et joint à tous les rapports, dossiers, mémoires.

Présidence de l'université
40 rue de rennes – BP 73532
49035 Angers cedex
Tél. 02 41 96 23 23 | Fax 02 41 96 23 00

université angers