Rapport de Stage

Licence Informatique $3^{\text{ème}}$ année

Tom Davot-Grangé

Dirigé par Olivier Baudon

Mai - Juin 2015

Table des matières

Introduction

Dans le cadre de ma licence Informatique à l'Université de Bordeaux, j'ai effectué mon stage au sein du Laboratoire Bordelais de Recherche Informatique (LaBRI) sous la direction d'Olivier Baudon, Maître de Conférence à l'Université de Bordeaux et membre du groupe de recherche Graphes et Applications. Ce groupe de recherche travaille sur la théorie des graphes et les problèmes classiques entourant celle-ci, notamment les problèmes liés aux colorations.

La théorie des graphes fut introduite par Léonard Euler au XVIIIe siècle dans un article traitant du problème des sept ponts de Königsberg, problème consistant à trouver un chemin passant par tous les ponts de la ville de Königsberg et revenant au point de départ du chemin sans passer deux fois par le même pont.

Mon travail consistait à implémenter en Java des algorithmes de coloration de graphes à partir d'articles de recherche. Une bibliothèque Java modélisant les graphes m'a été fournie par Olivier Baudon. Il existe de nombreux types de coloration, j'ai travaillé sur quatre d'entre elle : coloration propre des arêtes, bonne k-coloration des arêtes, coloration des sommets par pondération des arêtes et coloration des sommets par pondération des arêtes et des sommets.

J'ai également assisté, dans le cadre de ce stage, à deux groupes de travail portant sur des problèmes actuels de la théorie des graphes.

1 Algorithme de Mistra et Gries

Le premier algorithme que j'ai implémenté a été décrit dans l'article *A constructive proof of Vizing's Theorem*[1]. Cet algorithme se base sur le théorème de Vizing montrant qu'une coloration des arêtes d'un graphe peut s'effectuer avec un nombre de couleur compris entre Δ et $\Delta + 1$ où Δ est le degré maximum du graphe. L'algorithme propose une coloration en $\Delta+1$ couleurs. L'existence d'un algorithme polynomial garantissant une Δ-coloration si elle existe est liée à la question P=NP, l'existence d'une arête coloration d'un graphe en Δ couleurs étant un problème NP-complet[2].

L'algorithme permet de colorer une arête non colorée d'un graphe valide. Un graphe est dit valide s'il n'existe pas deux arêtes colorées incidentes à un même sommet qui possèdent la même couleur. Pour obtenir une coloration propre, il suffit de répéter l'algorithme sur un graphe jusqu'à ce que toutes les arêtes soient colorées. Nous qualifions une couleur *libre d'un sommet* si aucune des arêtes incidentes à ce sommet ne possède cette couleur.

1.1 Fan

L'algorithme utilise un structure de données appelée fan. Un fan $< f...l >$ de X, où X est un sommet, est une séquence de sommets qui satisfait les conditions suivantes (si u est un élément du fan, nous notons $u+$ son successeur dans le fan) :

— $< f...l >$ est une séquence non vide de sommets voisins de X
— l'arête Xf n'est pas colorée
— $\forall u \in < f...l >, u \neq l$, la couleur de l'arête $Xu+$ est *libre* de u

Exemple :

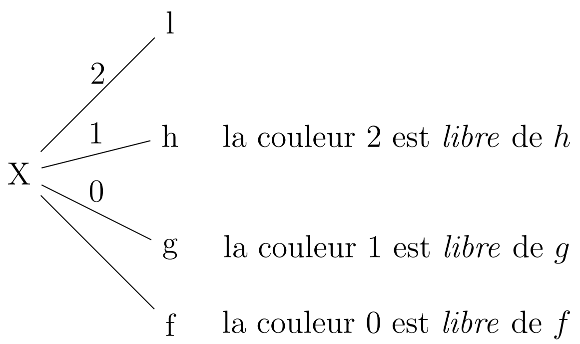


FIGURE 1 – La séquence $< f, g, h, l >$ est un fan de X

L'algorithme pour obtenir un fan maximal (i.e. qui ne peut pas être étendu) est le suivant :

Algorithme 1 : Construction d'un fan maximal d'un sommet X

creer_fan_maximal (G, X)
  Données :
  X : Sommet X
  G : graphe contenant X
  Résultat : F : fan maximal de X

  F $:= < f >$
  tant que $\exists u :: B$ **faire**
    ajouter u à la fin de F
  fin
  retourner F

B est le prédicat suivant :
$Xu \in E(G) \wedge u \notin F \wedge$ la couleur de Xu est *libre* du dernier sommet de F

1.2 Algorithme général

L'algorithme permettant de colorier une arête sans couleur est le suivant

Algorithme 2 : Coloration_arête

coloration_arete (G, X)
  Données :
  G : graphe à colorer
  X : sommet de G contenant une arête non colorée

  $F < f...l >:=$ creer_fan_maximal(G,X)
  Soit c une couleur *libre* de X
  Soit d une couleur *libre* de l telle que $c \neq d$
  Inverser_cdPath(G, X, c, d)
  Soit w un sommet qui satisfait :
  $w \in < f..l > \wedge < f...w >$ est un fan de $X \wedge d$ est *libre* de w
  Rotation$(G, X, < f...w >)$
  Colorer Xw en d

1.3 Inverser le cdPath

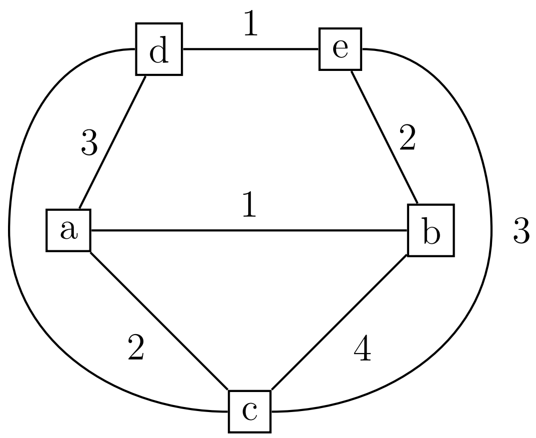
Le cdPath d'un sommet X est un chemin qui comprend des sommets liés entre eux alternativement par une arête de couleur c ou de couleur d. Ce chemin est unique et doit être maximum. Comme chaque sommet du graphe ne contient au maximum qu'une seule arête d'une même couleur, le chemin ne contient pas deux arêtes consécutives de même couleur. Ce chemin peut être vide. L'inversion du cdPath consiste à échanger les couleurs c en couleur d et les couleurs d en couleur c pour les arêtes du chemin.

1.4 Rotation

La rotation du fan $< f...w >$ consiste à donner à chaque arête Xu du fan la couleur de l'arête $Xu+$. L'arête w devient incolore.

1.5 Exemple

Nous allons dérouler l'algorithme sur le graphe suivant :



Le graphe est de degré 4, on peut donc colorier toutes arêtes avec au plus 5 couleurs différentes. L'arête qui lie les sommets d et c n'est pas colorée, on va appliquer l'algorithme à partir du sommet c afin de lui attribuer une couleur.

Création du fan du sommet c La première étape est de construire un fan maximal pour le sommet c, on applique l'algorithme 1 au sommet c :
$F := < d >$
La couleur de l'arête *ca* est *libre* du sommet d, on ajoute le sommet a au fan.
$F = < d, a >$
La couleur de l'arête *cb* est *libre* du sommet a, on ajoute le sommet b au fan.
$F = < d, a, b >$
La couleur de l'arête *ce* est *libre* du sommet b, on ajoute le sommet e au fan.
$F = < d, a, b, e >$

Tous les sommets voisins de c sont dans le fan.
On a trouvé un fan maximal F pour le sommet c : $< d, a, b, e >$

Inversion du cdPath
La couleur 1 est *libre* du sommet c, on prend $c = 1$;
La couleur 4 est *libre* du sommet e, on prend $d = 4$;
On inverse le cdPath (c, b, a) :

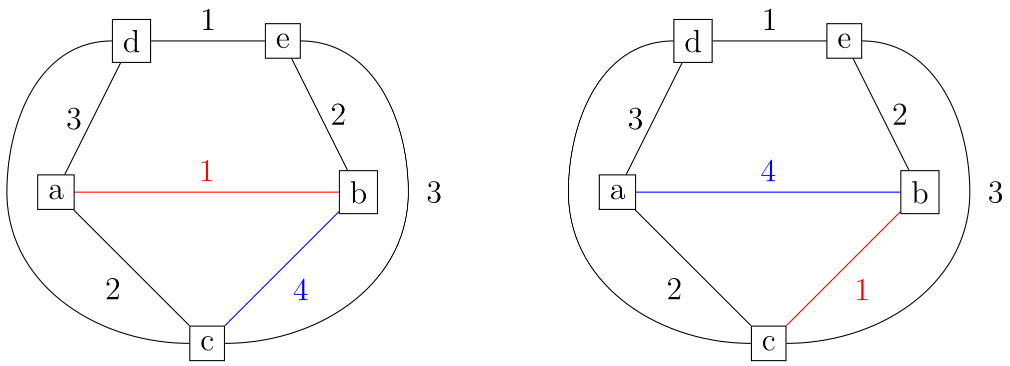


FIGURE 2 – Inversion du cdPath

Rotation
On cherche le premier sommet u du fan F tel que la couleur d soit *libre* de u :
Le sommet a est *libre* de la couleur d, on effectue la rotation sur le fan $< d, a >$

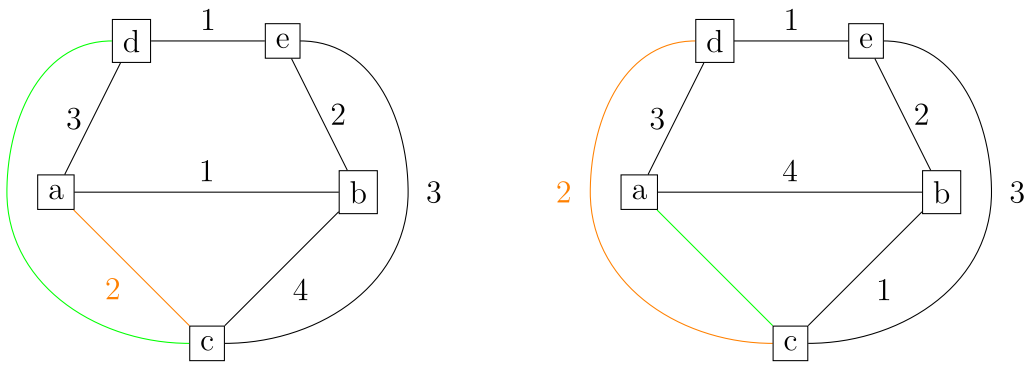
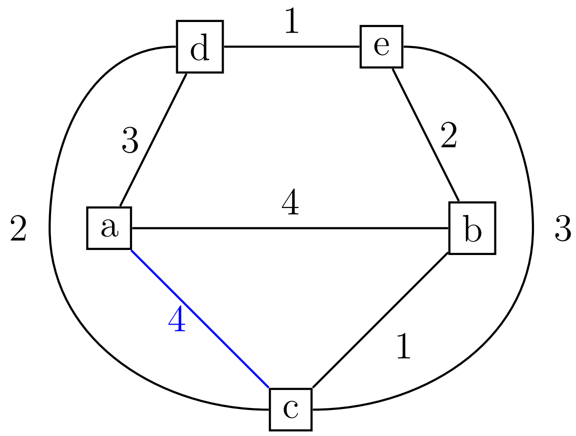


FIGURE 3 – Rotation

On donne ensuite la couleur d à l'arête ca :



Toutes les arêtes du graphes sont colorées et la coloration du graphe est propre.

2 Algorithme de bonne 2-coloration

L'algorithme de bonne 2-coloration suivant a été extrait de la démonstration de J.C. Fournier [3], l'algorithme n'est pas directement écrit dans l'article *Coloration des arêtes d'un graphe* [3], mais la forme de la démonstration permettait d'extraire facilement un algorithme.

2.1 Définition d'une bonne k-coloration

Pour un graphe G et C une coloration des arêtes de G en k couleurs, on note $\delta(x) = min\{d(x), k\} - |C_x|$ la déficience de C en un sommet x de G où $d(x)$ est le degré de x et C_x l'ensemble des couleurs des arêtes incidentes à x . On dit que C est une bonne k-coloration si pour tout sommets de G, la déficience de C est nulle. Ainsi une bonne k-coloration est une coloration où en chaque sommet le nombre de couleurs distinctes est maximum (mais la proportion de chaque couleur n'est pas forcément équilibrée).

Exemple :

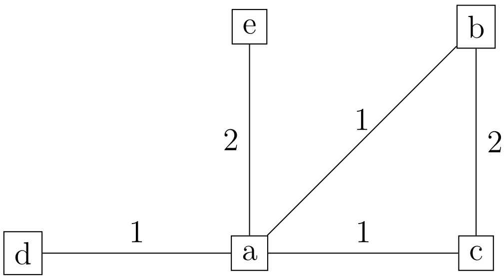


FIGURE 4 – Coloration 1

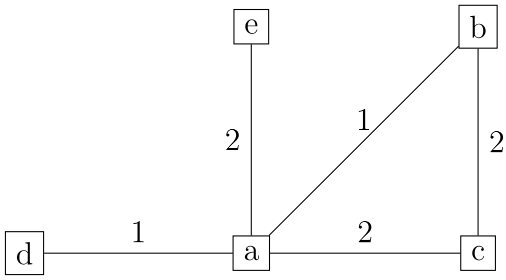


FIGURE 5 – Coloration 2

La coloration 1 est une bonne 2-coloration car tout sommet de degré 2 ou plus possède au moins une arête de chaque couleur. La coloration 2 est mauvaise car on a $\delta(c) = 1$.

2.2 Bicoloration

Le lemme de bicoloration s'énonce comme suit :

Lemme de bicoloration. *Tout graphe connexe différent d'un cycle impair (sans cordes) possède une bonne 2-coloration de ses arêtes.*

L'algorithme permettant d'obtenir une telle coloration est le suivant :

Algorithme 3 : Bonne 2-coloration

```
2_coloration (G, couleur_1, couleur_2)
    Données : G : graphe différent d'un cycle impair sans cordes
    couleur_1, couleur_2 : couleurs utilisées pour colorer les arêtes de G
    si ∀x ∈ V(G) , d(x) est pair alors
        xMax := sommetDegreMax(G)
        CE := cycleEulerien(G, xMax)
        couleur := couleur_1
        pour tous les e ∈ E(CE) faire
            C_e = couleur
            changerCouleur(couleur)
        fin
    fin
    sinon
        t := { x, x ∈ V(G) et d(x) est impair }
        tant que t ≠ ∅ faire
            extraire x de t
            colorier_Chaine(G, x, t, couleur_1)
        fin
        tant que ∃e ∈ E(G) ∉ C faire
            CC := {cycle d'arêtes incolores contenant e}
            couleur := couleur_1
            pour tous les e ∈ E(CC) faire
                C_e = couleur
                changerCouleur(couleur)
            fin
        fin
    fin
```

Algorithme 4 : colorer_chaine

colorer_chaine $(G, x, t, couleur)$

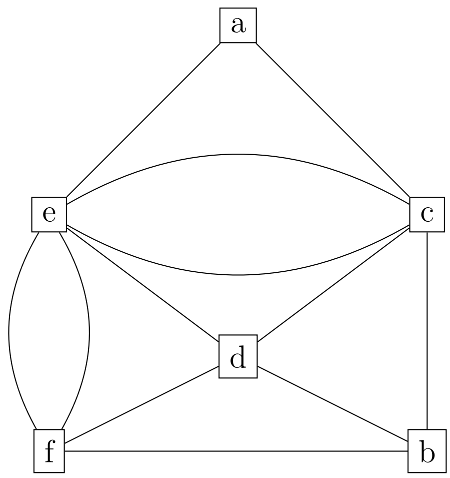
Données :
G : graphe contenant la chaîne à colorer
x : sommet où on se situe dans la chaîne
t : liste de sommets impairs de G
$couleur$: couleur que prendra la prochaine arête de la chaîne

si $\exists u \in N(x)$ *tel que* $ux \notin C$ **alors**
  $C_{ux} = couleur$
  colorer_chaine(G, u, t,changer_couleur($couleur$))
fin
sinon
  supprimer x de t
fin

L'algorithme va distinguer deux cas : si le graphe possède des sommets impairs ou non. Si le graphe ne possède que des sommets pairs, alors une coloration alternée des arêtes le long d'un cycle eulérien en partant d'un sommet ayant le degré maximum est une bonne 2-coloration. Sinon si le graphe possède $2p$ sommets impairs, la démonstration utilise la propriété qu'il existe une partition de G en p chaînes reliant 2 à 2 les sommets impairs. Ainsi une coloration alternée des arêtes de ces chaînes permet de créer une bonne 2-coloration. Pour cela l'algorithme va d'abord créer p chaînes avec la fonction *colorer_chaîne*. Cette fonction *colore* traverse les sommets et colore alternativement les arêtes par lesquelles elle passe, elle ne s'arrête que quand elle se trouve sur un sommet ne possédant plus aucune arête incolore incidente ; elle ne s'arrête donc que quand elle se trouve sur un sommet impair car si le sommet est pair elle peut ressortir de celui-ci par une autre arête que celle utilisée pour y accéder. Après avoir coloré ces chaînes, l'algorithme relie les sommets restants en cycles d'arêtes non colorées. Il est toujours possible de relier les arêtes restantes en cycle incolore : soit H, le sous-graphe de G ne contenant que les arêtes incolore, on distingue deux types de sommets : les sommets qui sont impairs dans G et ceux qui sont pairs dans G. On sait que la fonction *colorer_chaine* colorie deux arêtes incidentes à un sommet pair de G à chaque passage par ce sommet, les sommet pairs dans G restent donc pairs dans H. Pour les sommets impairs la fonction effectue un unique passage où soit elle ne fait que sortir de ce sommet (pour le premier sommet appelé) soit elle ne fait que rentrer dans ce sommet (pour le dernier sommet parcouru), les autres passages de la fonction s'effectuent comme pour un sommet pair. Le nombre d'arêtes colorées par *colorer_chaine* pour chaque sommet impair de G est donc impair, donc chaque sommet impair de G sera pair dans H. Donc tous les sommets de H sont pairs, il suffit donc d'extraire un cycle eulérien pour chaque composante connexe de H et de colorer alternativement les arêtes de ces cycles. Cela revient à insérer ces cycles dans une chaîne et permet à la partition de recouvrir totalement G.

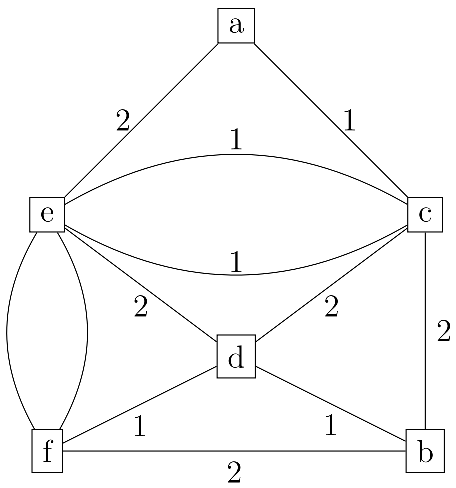
2.3 Exemple

Nous allons dérouler l'algorithme sur le graphe suivant :



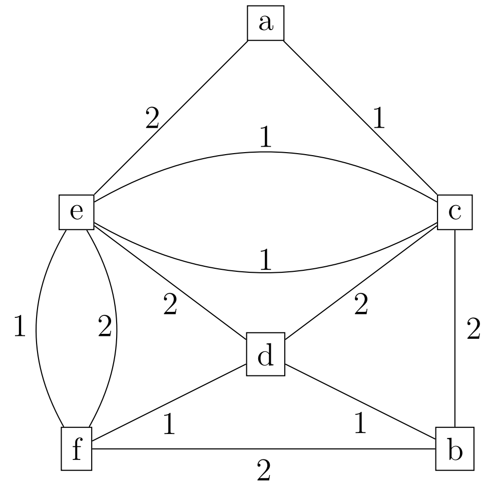
Le graphe contient des sommets impairs : t :=$\{c, b\}$
On supprimer le sommet c de t et on applique colorer_chaine à partir du sommet c.
L'arête ac n'est pas colorée : on lui applique la couleur 1 et on passe au sommet a.
L'arête ae n'est pas colorée : on lui applique la couleur 2 et on passe au sommet e.
L'arête ce n'est pas colorée : on lui applique la couleur 1 et on passe au sommet c.
L'arête bc n'est pas colorée : on lui applique la couleur 2 et on passe au sommet b.
L'arête bd n'est pas colorée : on lui applique la couleur 1 et on passe au sommet d.

L'arête cd n'est pas colorée : on lui applique la couleur 2 et on passe au sommet c.
L'arête ce n'est pas colorée : on lui applique la couleur 1 et on passe au sommet e.
L'arête de n'est pas colorée : on lui applique la couleur 2 et on passe au sommet d.
L'arête df n'est pas colorée : on lui applique la couleur 1 et on passe au sommet f.
L'arête bf n'est pas colorée : on lui applique la couleur 2 et on passe au sommet b.
Toutes les arêtes incidentes au sommet b sont colorées, on supprime b de t.
t est vide.
On obtient le graphe intermédiaire suivant :



L'arête ef n'est pas colorée, on récupère le cycle contenant ef : $CC := \{ef, fe\}$
On colorie alternativement les arêtes :
ef prend la couleur 1.
fe prend la couleur 2.
Toutes les arêtes du graphe sont colorées.

On obtient donc la coloration suivante :



On constate que pour chaque sommet les deux couleurs apparaissent, cette 2-coloration est donc une bonne coloration.

3 h-coloration

Cet algorithme est également extrait de l'article *Coloration des arêtes d'un graphe* [3] à partir des démonstration des théorèmes 1 et 2 de cet article.

3.1 Théorème 1

Théorème 1. *Un graphe simple de degré maximum h admet une h-coloration des arêtes du graphe telle que :*

$$\delta(x) \begin{cases} = 0 & si\ d(x) < h \\ \leq 1 & si\ d(x) = h \end{cases}$$

3.2 Algorithme

Soit G un graphe possédant une coloration sans propriété particulière. L'algorithme permettant de trouver une h-coloration de G telle que celle décrite dans le théorème 1 est le suivant :

Algorithme 5 : h-coloration du théorème 1

```
h-coloration (G)
    Données : G : graphe de degré maximum h ayant déjà une
              coloration de ses arêtes
    tant que ∃x ∈ V(G) tel que
    (δ(x) > 0 ∧ d(x) < h) ∨ (δ(x) > 1 ∧ d(x) = h) faire
        recolorer(G, x)
    fin
```

Algorithme 6 : recolorer

recolorer (G, x)

Données :

G : graphe de degré maximum h

x : sommet de G à modifier

$\alpha :=$ couleur $\in C_x$, $|C_x(\alpha)| > 1$

$\beta :=$ couleur $\notin C_x$

$H := \{$composante connexe contenant x du sous-graphe engendré par les arêtes de couleur α et $\beta\}$

si *H n'est pas un cycle impair sans cordes* **alors**

  2_coloration(H, α, β)

fin

sinon

  $y :=$ voisin de x dans H

  si *$\nexists$ couleur $\alpha_2 \notin C_y$* **alors**

    $C(xy) = \beta$

  fin

  sinon

    $C(xy) = \alpha_2$

    si $|C_x(\alpha_2)| > 1$ **alors**

      recolorer(G, x)

    fin

  fin

fin

h-coloration L'algorithme se décompose en deux fonctions : *h-coloration* et *recolorer*. *h-coloration* parcours les sommets de G et appelle *recolorer* pour chaque sommet ne respectant pas les propriétés du théorème 1. Cette décomposition en deux fonctions permet à *recolorer* de s'appeler récursivement.

recolorer Si un sommet x_0 possède une déficience trop forte cela signifie qu'une couleur α apparaît au moins deux fois à ce sommet et une autre couleur β pas du tout. Si le sous-graphe induit par les arêtes de couleur α et β n'est pas un cycle impair, une simple 2-coloration avec les couleurs α et β permet de réduire la déficience en x_0. Sinon si le voisin y de x_0 dans H ne possède pas de couleur $\alpha_2 \notin |C_y|$, alors $d(y) = h$ et la recoloration de l'arête x_0y en β permet de faire baisser la déficience de x_0 et respecte la coloration demandée car $\delta(y) <= 1$. Si α_2 existe, alors l'algorithme colore l'arête x_0y en α_2 et recommence l'opération jusqu'à ce qu'il trouve un H

différent d'un cycle impair. Si l'algorithme utilise une couleur α qu'il a déjà utilisée pour un même sommet x_0, alors on a $H_n = H_\gamma$ - $\{xy_\gamma\}$ où H_n, H_γ et y_γ représentent respectivement le sous-graphe H de l'instance actuelle de la fonction *recolorer*, le sous-graphe H et le sommet de voisin y utilisés dans un précédent appel à la fonction *recolorer*. Or comme l'arête xy_γ du cycle impair H_γ a été recolorée, elle ne fait pas parti de H_n donc H_n est forcément différent d'un cycle impair.

3.3 Diminution de la déficience totale

On peut modifier l'algorithme de h-coloration afin d'obtenir une bonne h-coloration pour les graphes ne possédant pas de cycle de sommets de degré maximum :

Théorème 1. *Soit G un graphe simple de degré h. Si G ne possède pas de cycle de sommets de degré h, alors $\chi'(G) = h$.*

où $\chi'(G)$ est l'indice chromatique de G. Dans la boucle de la ligne 2, pour les sommets de degré maximum, il faut appeler la fonction suivante à la place de recolorer :

Algorithme 7 : recolorer pour les sommets de degré maximum

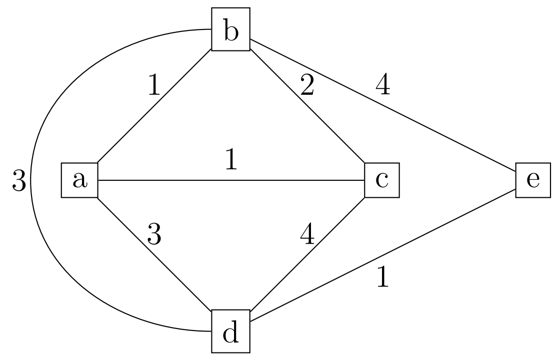
recolorer_sommet_Max $(G, x, chemin)$
    Données :
    G : graphe à colorer
    x : sommet de G à modifier
    chemin : tableau contenant les sommets modifiés par cette fonction
    si $x \notin chemin$ **alors**
        $\alpha :=$ couleur $\in C_x$, $|C_x(\alpha)| > 1$
        $\beta :=$ couleur $\notin C_x$
        $H :=$ {composante connexe contenant x du sous-graphe engendré par les arêtes de couleur α et β}
        si *H n'est pas un cycle impair sans cordes* **alors**
            2_coloration(H)
        fin
        sinon
            $y :=$ voisin de x dans $H \notin chemin$
            si $\exists$ *couleur* $\alpha_2 \notin C_y$ **alors**
                $C(xy) = \alpha_2$
                si $|C_x(\alpha_2)| > 1$ **alors**
                    recolorer_sommet_Max$(G, y, chemin)$
                fin
            fin
            sinon
                $C(xy) = \alpha_2$
                si $|C_x(\alpha_2)| > 1$ **alors**
                    $chemin = \{chemin, x\}$
                    recolorer_sommet_Max$(G, x, chemin)$
                fin
            fin
        fin
    fin

La différence avec la fonction *recolorer* est que si y est un sommet de degré maximum (dans le cas où la couleur α_2 n'existe pas), alors l'appel récursif de recolorer_sommet_Max s'effectuera sur y. Si la fonction retombe sur un sommet qu'elle a déjà traité, cela signifie que le graphe contient un cycle de sommets de degré maximum et que la déficience d'un des sommets de ce cycle doit être égale à 1.

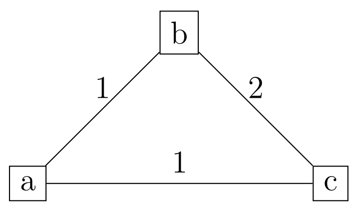
Soit G un graphe contenant c cycles de sommets de degré maximum, en appliquant cet algorithme, on obtient une coloration telle que : $\sum \delta(G) \leq c$

3.4 Exemple

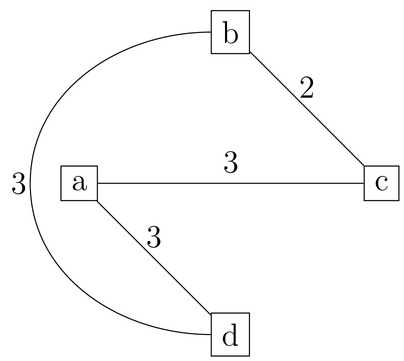
Nous allons appliquer l'algorithme sur le graphe suivant :



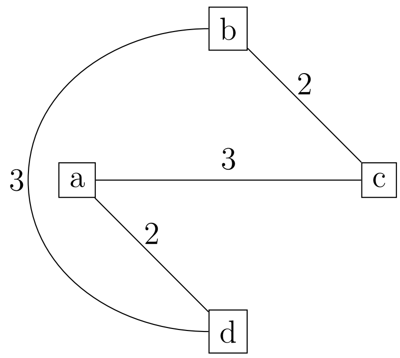
Il s'agit d'un graphe de degré maximum 4. Comme celui-ci ne contient pas de cycle de sommets de degré 4, nous pouvons obtenir une bonne 4-coloration. La déficience du sommet a n'est pas nulle, on appel donc la fonction *recolorer* sur ce sommet. On prend $\alpha = 1$ et $\beta = 2$, on obtient H :



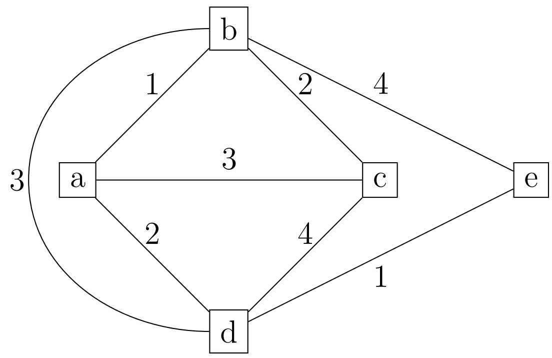
Comme H est un cycle impair sans cordes, on ne peut pas faire une bonne 2-coloration. On prend $\alpha_2 = 3$, et on colore l'arête ac avec cette couleur. Comme la couleur $\alpha_2 = 3$ apparaît deux fois en a, on appelle à nouveau la fonction *recolorer* sur a. On prend $\alpha = 3$ et $\beta = 2$, on obtient H :



Cette fois-ci, H n'est pas un cycle impair, on peut donc effectuer une bonne 2-coloration sur H avec les couleurs α et β :



Tous les sommets possède une déficience nulle, on obtient la coloration suivante :



Aucun des sommets de G ne possède deux arêtes incidentes de même couleur, on obtient bien une bonne 4-coloration.

4 Coloration des sommets par pondération des arêtes

L'algorithme suivant est issu de l'article *Vertex-coloring edge-weightings : Towards the 1-2-3-conjecture*[4]. Cette fois-ci le but de cette coloration est d'attribuer un poids à chaque arête d'un graphe simple G de façon à ce que pour tous les sommets v de G, la somme des poids des arêtes incidentes à v soit différentes de celle des voisins de v. Cette somme est donc une coloration propre des sommets de G. L'article montre qu'il est possible d'obtenir une telle coloration en utilisant un ensemble de cinq valeurs pour pondérer les arêtes sauf si G possède une arête isolée (dans ce cas aucune coloration propre par somme n'est possible).

4.1 Ordonnancement des sommets

La première étape de l'algorithme consiste à ordonner les sommets du graphe $V(G) = \{v_1, v_2, ..., v_n\}$ de façon à ce que $d(v_n) \geq 2$ et que $\forall v_i \in V(G) - \{v_n\}, \exists v_j \in \{v_{i+1}, v_{i+2}, ..., v_n\}, v_j \in N(v_i)$.

Algorithme 8 : Ordonnancement des sommets

ordonner_sommets (G)
    Données : G : graphe dont on veut ordonner les sommets
    Résultat : *ordre* : tableau contenant les sommets ordonnés
    $File$:= file contenant un sommet de G dont le degré est supérieur ou égal à 2
    tant que *$File$ n'est pas vide* **faire**
        v = défiler($File$)
        ajouter v au début de *ordre*
        pour tous les $n \in N(v)$ **faire**
            enfiler($File, n$)
        fin
    fin

Cet algorithme est en fait un simple parcours en largeur des sommets de G en partant d'un sommet de degré supérieur ou égal à 2.

4.2 Attribution d'un ensemble de couleur

Pour cette partie $f(v)$ désigne la somme des poids d'arêtes incidentes au sommet v et $f(e)$ désigne le poids de l'arête e. L'algorithme va parcourir les sommets de G dans l'ordre $\{v_0, v_1, ..., v_n\}$ établi avec l'algorithme 8. Pour chacun des sommets $v_k, 0 \leq k < n$, il va lui attribuer un ensemble de deux couleurs $W(v_k) = \{w, w+2\}$ tel que $f(v_k) \in W(v_k)$ et que $w \bmod(4) \in \{1, 2\}$. Il faut attribuer W de façon à ce que $\forall v_i \in N(v_k), 0 \leq i < k, W(v_i) \cap W(v_k) = \emptyset$. L'algorithme utilisé ici ne va pas à proprement parler attribuer un ensemble de couleur à chaque sommet, mais va utiliser une variable $position_k$ qui indique si $f(v_k)$ possède la valeur de w ou de $w+2$. Ainsi il suffit d'ajouter ou de retirer 2 à $f(v_k)$ pour obtenir l'autre couleur de $W(v)$. L'attribution de $W(v_k)$ s'effectue en faisant varier le poids des arêtes $\{v_i v_k, 0 \leq i < k\}$ de façon suivante :

— l'arête peut garder sa valeur initiale
— si $position_i = 0$, $f(v_i v_k) := f(v_i v_k) + 2$
— si $position_i = 2$, $f(v_i v_k) := f(v_i v_k) - 2$

On modifie la valeur d'une arête $v_j v_k$,
$j = min\{j > k, v_j \in N(v_k)\}$ de telle façon que :

— si $f(v_k) \bmod(4) \in \{1, 2\}$, $f(v_j v_k) \in \{2, 3\}$
— sinon $f(v_j v_k) \in \{3, 4\}$

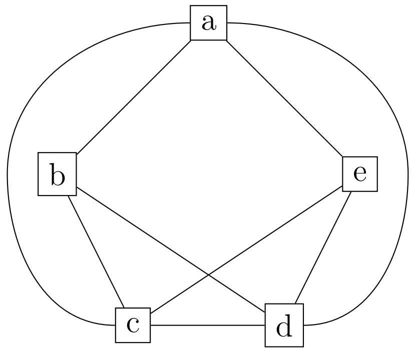
Cela laisse un intervalle de $2d + 3$ valeurs pour $f(v_k)$ où d est le nombre de voisins de v_k déjà parcourus. Comme chacun des ces voisins ne peut bloquer en tout que deux valeurs de cet intervalle et que la condition sur la valeur de $f(v_j v_k)$ ne peut bloquer que les valeurs maximum et minimum de cet intervalle, on peut être sûr qu'il existe au moins une valeur possible pour $f(v_k)$. Si $f(v_k) \bmod(4) \in \{1, 2\}$ alors $W(v_k) = \{f(v_k), f(v_k) + 2\}$, sinon $W(v_k) = \{f(v_k) - 1, f(v_k)\}$.

4.3 Attribution de la couleur du dernier sommet

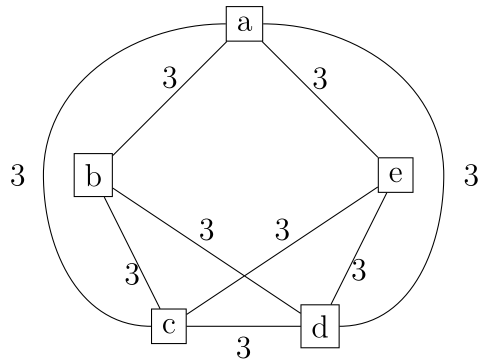
Pour attribuer une couleur au dernier sommet v_n, on peut faire varier la valeur des arêtes incidentes à v_n de la même façon que précédemment, ce qui laisse un intervalle de $d(v_n) + 1$ valeurs possibles. Soit a la plus petite valeur de cet intervalle. Si $a \bmod(4) \in \{2, 3\}$ alors donner la valeur minimale à chacune des arêtes incidentes à v_n permet d'obtenir une coloration propre. Sinon, s'il existe $v_i \in N(v_n), f(v_i) \neq f(v_n)$, alors donner la plus haute valeur possible à l'arête $v_i v_n$ et la plus basse valeur pour toutes les autres permet d'obtenir une coloration propre. Sinon, il faut donner la plus haute valeur pour deux arêtes et la plus basse valeur pour toutes les autres.

4.4 Exemple

Nous allons dérouler l'algorithme sur le graphe suivant :

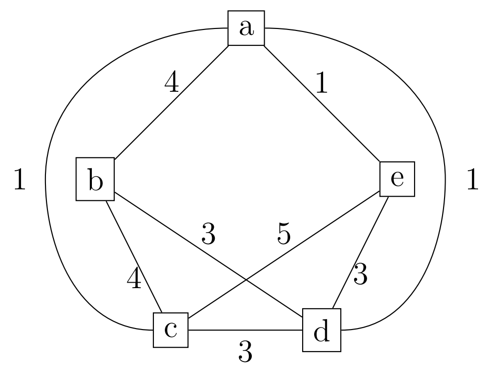


Avec un parcours en largeur, on va déterminer dans quel ordre l'algorithme va parcourir les sommets. On peut commencer ce parcours par le sommet a car le degré de celui-ci est supérieur ou égale à 2. On obtient cet ordre de parcours : $\{e, d, c, b, a\}$. On commence par initialiser tous les poids des arêtes à 3.



Puis on parcours tous les sommets dans l'ordre $\{e, d, c, b, a\}$.
Sommet e : on a $f(e) = 9$, on lui attribue l'ensemble $W(e) = \{9, 11\}$ et $position_e = 1$.
Sommet d : on a $f(d) = 12$, on lui attribue l'ensemble $W(d) = \{10, 12\}$ et $position_d = 2$.
Sommet c : on a $f(c) = 12 \in W(d)$, on doit modifier la valeur des arêtes incidentes à c. On prend $f(ce) = 5$ ($position_e = 2$) et $f(cb) = 4$. On lui attribue l'ensemble $W(c) = \{13, 15\}$ et $position_c = 2$.
Sommet b : on a $f(b) = 10 \in W(d)$, on doit modifier la valeur des arêtes

incidentes à b. On prend $f(ba) = 4$. On lui attribue l'ensemble $W(b) = \{9, 11\}$ et $position_b = 2$.
On arrive au dernier sommet a : on met toutes arêtes incidente à a à leur plus basse valeur possible : $f(ae) = 1$, $f(ad) = 1$, $f(ab) = 2$ et $f(ac) = 1$ on obtient alors $f(a)\ mod(4) = 5\ mod(4) = 1 \in \{0, 1\}$. On a $f(b) \neq f(a)$, on met donc la plus haute valeur possible sur l'arête ab et on laisse la plus petite valeur pour les autres. On obtient la finale coloration suivante :



5 Coloration des sommets par pondération des arêtes et de sommets

Pour cet algorithme nous allons attribuer un poids aux arêtes et aux sommets, une coloration sera propre si pour chaque sommet v d'un graphe G, la somme des poids des arêtes incidentes à v et du poids de v est différente de celle de ces voisins. L'article *A note on 1-2-conjecture* [5] montre qu'on peut obtenir une coloration propre en utilisant l'ensemble $\{1, 2\}$ pour pondérer les sommets et l'ensemble $\{1, 2, 3\}$ pour pondérer les arêtes.

5.1 Algorithme

Soit $\psi(v) = w(v) + \sum_{n \in N(v)} w(vn)$, où v est un sommet de G et w est le poids d'une arête ou d'un sommet. Cet algorithme ressemble beaucoup à celui de la conjecture 1-2-3. La première étape de l'algorithme consiste à initialiser tous les poids des arêtes de G à 2 et tous les poids des sommets à 1, on va également utiliser une variable intermédiaire ϕ qui sera initialisée lorsque l'algorithme a fini de parcourir pour la première fois un sommet. Soit $\{v_0, v_1, ..., v_n\}$ l'ordre dans lequel les sommets de G vont être parcourus, l'ordre n'a ici pas d'importance. Lorsqu'on parcours le sommet v_k, $0 \leq i \leq n$, si $\psi(v_k) = \phi(v_i), 0 \leq i < k$, on peut modifier les arêtes $v_k v_i, 0 \leq i < k$ de la façon suivante :

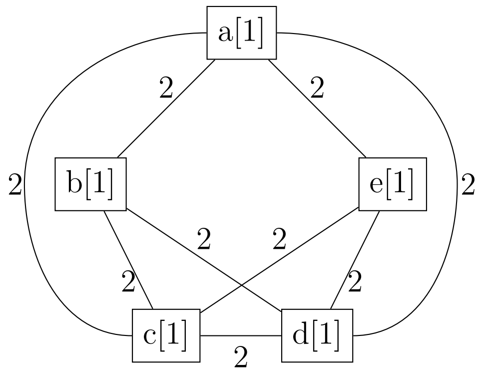
— Le poids de l'arête peut rester inchangé
— $w(v_k v_i) = w(v_k v_i) - 1$ si $\psi(v_i) = \phi(v_i)$
— $w(v_k v_i) = w(v_k v_i) + 1$ sinon

Cela laisse un intervalle de $d+1$ valeurs possibles pour ψ où d est le nombre de sommets voisins de v_k déjà parcourus par l'algorithme. Sachant que chacun de ces sommets ne peut bloquer qu'une seule valeur de cet intervalle, on est sûr qu'il reste au moins une valeur correcte possible. Finalement on attribue la valeur de $\psi(v_k)$ à $\phi(v_k)$.

Quand tous les sommets de G ont été parcourus, pour chacun des sommets v de G tels que $\psi(v) = \phi(v) - 1$ on ajoute 1 à la valeur de $w(v)$.

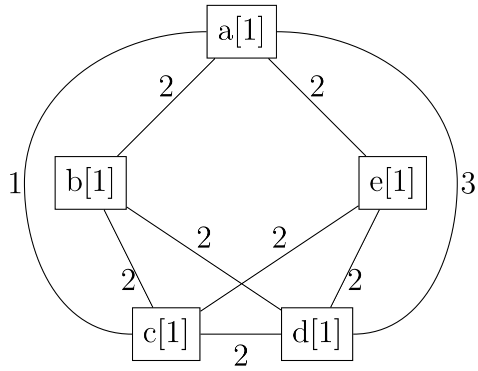
5.2 Exemple

On utilise la même graphe que dans l'exemple précédent, on initialise le poids de toutes les arêtes à 2 et les poids de tous les sommets à 1.



On va parcourir les sommets dans l'ordre alphabétique. Sommet a : $\phi(a) =$ 9.
Sommet b : $\psi(b) = 7$. On attribue $\phi(b) = 7$.
Sommet c : $\psi(c) = \phi(a)$. On prend $w(ac) = 1$. On attribue $\phi(c) = 8$.
Sommet d : $\psi(d) = \phi(a)$. On prend $w(ad) = 3$. On attribue $\phi(d) = 10$.
Sommet e : $\psi(d) = 7$. On attribue $\phi(d) = 7$.
On attribue maintenant les poids des sommets :
Sommet a : $\psi(a) = \phi(a)$. On ne change rien.
Sommet b : $\psi(b) = \phi(b)$. On ne change rien.

Sommet c : $\psi(c) = \phi(c)$. On ne change rien.
Sommet d : $\psi(d) = \phi(d)$. On ne change rien.
Sommet e : $\psi(e) = \phi(e)$. On ne change rien.
On obtient la coloration finale suivante :



6 Bilan du stage

Tous les algorithmes proposés ont pu être implémentés pendant la durée du stage. Des tests ont également pendant la durée du stage. Tout d'abord à l'aide de classes de test, puis à la demande de mon encadrant, certains d'entre eux ont été réécrits à l'aide de l'outil JUnit.

Les premiers jours du stage ont davantage été consacrés à la découverte de la bibliothèque Java avec laquelle j'ai travaillé pour implémenter les algorithmes qu'à l'implémentation proprement dite des algorithmes. Cela n'a pas été particulièrement compliqué, mais ayant effectué mes deux premières années de licence à l'Université Claude Bernard Lyon 1, je n'avait jamais fait de Java, il m'a fallu quelques temps avant d'être à l'aise avec le langage et la bibliothèque.

J'ai rencontré deux principales difficultés lors de ce stage. La première a été de comprendre les articles dont les algorithmes que j'ai implémentés étaient issus, la plupart des erreurs survenues étaient dues au fait que j'avais mal compris un passage d'un article. L'article sur la conjecture 1-2-3[4] a été le plus difficile à comprendre, d'une part l'algorithme n'était pas écrit explicitement et d'autre part la notation utilisée était assez compliquée. La deuxième difficulté était de trouver un algorithme pour certaines choses qui étaient triviales pour les auteurs des articles mais pas pour moi. Par exemple pour l'algorithme permettant de trouver une bonne 2-coloration d'un graphe, il est écrit dans l'article qu'il existe une partition d'un graphe en p graphe partiels si celui-ci contient $2p$ sommets impairs ; j'ai dû trouver moi-même l'algorithme permettant d'effectuer une telle partition du graphe.

Je pense que ce stage a été très enrichissant, d'une part parce qu'il m'a fait travailler en Java ce que je n'avais jamais fait ; j'aurai besoin de connaître ce langage pour suivre les cours de master. D'autre part, travailler sur des articles de recherche m'a permis de me familiariser avec les démonstrations d'algorithmes tout en me laissant un certain travail de recherche de solution à effectuer lors de l'implémentation des algorithmes lorsque la démonstration restait très générale.

Références

[1] J.Mistra, D. Gries, *A constructive proof of Vizing's Theorem*. Information Processing Letter 41, 1992, 131-133.

[2] M.R. Garey, D.S. Johnson, *Computer and Intractability : A guide to the Theory of NP-Completeness*. Freeman and Compagny, 1979.

[3] J.C. Fournier, *Coloration des arêtes d'un graphe*. Cahiers CERO (Bruxelles) 15, 1973, 311–314.

[4] M.Kalkowski, M. Karoński, F. Pfender, *Vertex-coloring edge-weightings : Towards the 1-2-3-conjecture*. Journal of Combinatorial Theory, Series B, 2010 347-349.

[5] M.Kalkowski, *A note on 1-2-conjecture*, Rapport de Recherche.