

ESIREM

4A INFOTRONIQUE MODULE ITIS42

COMPTE-RENDU DE TRAVAUX PRATIQUES

Ingénierie des systèmes d'information

Vincent CANDAPPANE

30 avril 2018



Table des matières

| | | |
|----------|---------------------------------------------------------------------------------|-----------|
| | Introduction générale | 1 |
| 1 | Étude réalisée | 2 |
| 1.1 | Graphes hamiltoniens | 2 |
| 1.2 | Algorithme de coloriage | 2 |
| 2 | Problème : Une histoire de Bowling | 3 |
| 2.1 | Énoncé | 3 |
| 2.2 | Problème | 3 |
| 3 | Modélisation via les graphes | 4 |
| 3.1 | Tableau matriciel d'incompatibilité | 4 |
| 3.2 | Graphe d'incompatibilité | 4 |
| 4 | Programme linéaire | 5 |
| 4.1 | Approche 'naïve' : | 5 |
| 4.2 | Approche 'mathématique' : | 5 |
| 4.3 | Approche et solution finale : | 5 |
| 4.3.1 | Vérification d'une chaîne eulérienne | 6 |
| 4.3.2 | Appliquons l'algorithme de Welsh et Powell (optimisé à notre situation) : | 6 |
| 5 | Programmation informatique en Java | 8 |
| 6 | Résultat | 10 |
| 7 | Performance | 11 |
| 8 | Conclusion générale | 12 |
| | Sitographie | 12 |
| 9 | Annexe | 13 |

Table des figures

| | | |
|-----|----------------------------------------------------------------------------------|----|
| 3.1 | Tableau matriciel d'incompatibilité d'humeur | 4 |
| 3.2 | Graphe d'incompatibilité | 4 |
| 4.1 | Tableau des couleurs | 6 |
| 4.2 | Coloration du graphe selon une variante de l'algorithme de Welsh-Powel | 7 |
| 4.3 | Résultat final du groupe à inviter | 7 |
| 5.1 | Schéma fonctionnel de l'algorithme | 8 |
| 5.2 | Implémentation de la matrice sous Java | 9 |
| 6.1 | Simulation avec un graphe prédéfini à 11 sommets sous eclipse | 10 |
| 6.2 | Résultat final du groupe à inviter | 10 |
| 6.3 | Simulation avec un graphe aléatoire à 5000 sommets sous eclipse | 10 |

Introduction générale

Dans le cadre de nos travaux pratiques de l'ingénierie des systèmes d'information au sein de l'ESIREM, nous allons appliquer et résumer les connaissances acquises au cours du module ITIS42.

Ce rapport retrace notre démarche et notre capacité à résoudre une situation délicate par l'intermédiaire des graphes et de la recherche opérationnelle.

L'objectif de ce TP est de pouvoir réaliser à partir d'un graphe, l'invitation du plus grand nombre possible de personnes capables de se supporter entre elles parmi les collègues de Christophe. En réalisant cela nous aideront Christophe à organiser sa soirée bowling de manière optimisée.

1. Étude réalisée

1.1 Graphes hamiltoniens

On appelle cycle hamiltonien d'un graphe G un cycle passant une et une seule fois par chacun des sommets de G . Un graphe est dit hamiltonien s'il possède un cycle hamiltonien. On appelle chaîne hamiltonienne d'un graphe G une chaîne passant une et une seule fois par chacun des sommets de G . Un graphe ne possédant que des chaînes hamiltoniennes est semi-hamiltonien. Contrairement aux graphes eulériens, il n'existe pas de caractérisation simple des graphes (semi-)hamiltoniens. On peut énoncer quelques propriétés et conditions suffisantes :

- un graphe possédant un sommet de degré 1 ne peut pas être hamiltonien ;
- si un sommet dans un graphe est de degré 2, alors les deux arêtes incidentes à ce sommet doivent faire partie du cycle hamiltonien ;
- les graphes complets K_n sont hamiltoniens.

Théorème d'Ore

Soit G un graphe simple d'ordre $n \geq 3$. Si pour toute paire $\{x, y\}$ de sommets non adjacents, on a $d(x) + d(y) \geq n$, alors G est hamiltonien.

Théorème de Dirac

Soit G un graphe simple d'ordre $n \geq 3$. Si pour tout sommet x de G , on a $d(x) \geq \frac{n}{2}$, alors G est hamiltonien.

En effet, un tel graphe vérifie les conditions du théorème précédent, car si x et y ne sont pas adjacents, on a bien : $d(x) + d(y) \geq \frac{n}{2} + \frac{n}{2} = n$.

1.2 Algorithme de coloriage

Définition

- Une coloration d'un graphe consiste en l'attribution de couleurs aux sommets, de telle sorte que deux sommets adjacents n'aient jamais la même couleur.
- Le nombre chromatique $\chi(G)$ d'un graphe G est le nombre minimum de couleurs nécessaires à sa coloration, c'est-à-dire, le nombre de couleurs minimum pour colorier les sommets du graphe sans que deux sommets adjacents ne soient de la même couleur.

Propriété : Nombre chromatique

- Le nombre chromatique $\chi(G)$ d'un graphe complet d'ordre n est n .
- Le nombre chromatique $\chi(G)$ d'un graphe est supérieur ou égal à celui de chacun de ses sous-graphes.
- Le nombre chromatique $\chi(G)$ d'un graphe G est inférieur ou égal à $\Delta + 1$, où Δ est le plus grand degré des sommets.

2. Problème : Une histoire de Bowling

2.1 Énoncé

Christophe veut organiser une grande soirée bowling avec ses collègues de travail. En proposant son idée à ses différents collègues, il découvre les inimitiés qu'il peut y avoir entre collègues.

Soucieux de regrouper le plus grand nombre de personnes qui ne se détestent pas entre elles, il demande à chaque personne la liste des personnes qu'elle ne souhaite pas voir, et invitera le plus grand nombre possible de personnes qui sont capables de se supporter entre elles. Pouvez-vous l'aider ?

2.2 Problème

Afin de concrétiser la situation, nous allons nous baser sur les critères suivants : onze jeunes collègues de travail sont invités pour la soirée bowling de Christophe. Cependant, nous sommes face à une situation d'incompatibilité d'humeur entre collègues.

L'organisateur, ici Christophe, soucieux d'éviter les problèmes, veut tenir compte des inimitiés entre ces collègues qui sont les suivantes :

- Alex est le plus difficile, il ne peut supporter les collègues suivant : Benjamin, Cyril, Damien, Emmanuel, Grégoire, Inès, Jacob et Kevin ;
- Benjamin n'apprécie pas Alex et Cyril ;
- Cyril refuse de discuter avec Alex, Benjamin et Damien ;
- Damien ne supporte pas Alex, Cyril, Emmanuel et Franck ;
- Emmanuel ne veut côtoyer ni Alex, Damien, Franck et Grégoire ;
- Frank n'apprécie pas Damien, Emmanuel et Grégoire ;
- Grégoire ne veut ni voir Alex, ni Emmanuel, ni Franck, ni Hector, ni Inès et ni Kevin ;
- Hector est le moins difficile, il aime tout le monde sauf Grégoire à cause d'une vieille idylle amoureuse, et d'un désaccord sur la notion de consentement ;
- Inès ne viendra que si Alex, Grégoire, Jacob et Kevin ne viennent pas, car selon lui, ce sont des cons ;
- Jacob ne veut pas voir Alex et Inès, car selon lui, "ils sont débiles, mais débiles... je ne savais pas qu'on pouvait être aussi débile que ça" ;
- Kevin ne veut ni voir Alex, car c'est son $N + 1$, ni Grégoire, car selon lui aussi, Grégoire est un con. De plus, il ne possède pas une bonne relation avec Inès.

3. Modélisation via les graphes

3.1 Tableau matriciel d'incompatibilité

Afin de simplifier ces données, nous allons tout simplement dresser un tableau matriciel définissant les incompatibilités d'humeur entre individus. Il s'agit d'un tableau avec tous les noms en abscisse, mais aussi en ordonnée. La présence d'une croix signifie que les collègues ne peuvent pas se supporter entre elles, dans le cas échéant, la case sera vide. Les cases rouges s'agit des liaisons interdites, car l'on considère que chaque personne peut se supporter avec eux-même.

| | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|
| A | | X | X | X | X | | X | | X | X | X |
| B | X | | X | | | | | | | | |
| C | X | X | | X | | | | | | | |
| D | X | | X | | X | X | | | | | |
| E | X | | | X | | X | X | | | | |
| F | | | | X | X | | X | | | | |
| G | X | | | | X | X | | X | X | | X |
| H | | | | | | | X | | | | |
| I | X | | | | | | X | | | X | X |
| J | X | | | | | | | | X | | |
| K | X | | | | | | X | | X | | |

FIGURE 3.1 – Tableau matriciel d'incompatibilité d'humeur

3.2 Graphe d'incompatibilité

L'idée ici est de relier par une arête les sommets représentant les individus incompatibles, le graphe non-orienté obtenu est appelé graphe d'incompatibilité traduisant ainsi notre situation :

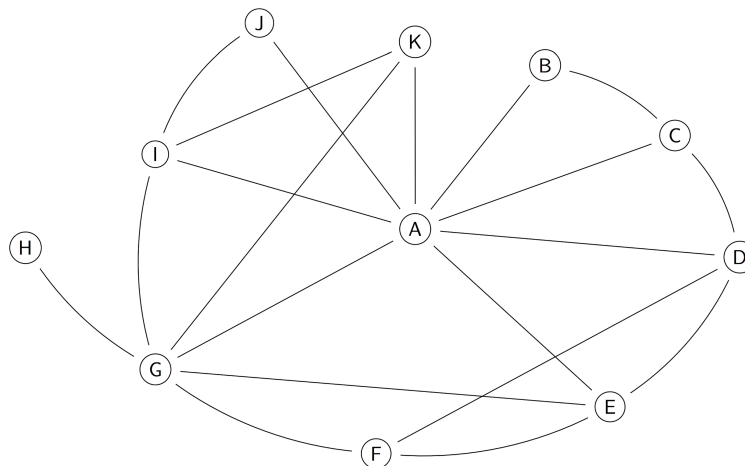


FIGURE 3.2 – Graphe d'incompatibilité

Les sommets représentent les amis de Christophe (chaque sommet sera noté par la première lettre du prénom). Les arêtes représentent les incompatibilités d'humeur entre deux « amis ». Bien entendu la liaison d'une personne à une autre est bidirectionnelle, ce qui veut dire que si une personne A ne s'entend pas bien avec une personne B, l'inverse (corollaire) sera valide aussi.

4. Programme linéaire

4.1 Approche 'naïve' :

L'algorithme naïf parcourt les sommets du graphe aléatoirement (sans ordre précis). Pour chacun d'eux, il établit la liste des couleurs de ses voisins et donne au sommet courant la plus petite couleur possible non attribuée à l'un d'eux. Cet algorithme a pour avantage d'avoir un temps d'exécution rapide, mais ne donnera pas forcément un résultat optimal.

Pour des raisons d'optimisation, cette approche sera implémentée sous forme de code, elle permettra de réaliser la comparaison avec la solution finale, et d'évaluer leurs capacités. Afin de pouvoir développer nos idées, nous allons avoir recours aux propriétés du graphe pour améliorer l'algorithme recherché. Quel type de graphe envisager ?

4.2 Approche 'mathématique' :

Ici, une solution un peu moins empirique peut être trouvée : afin de créer des groupes d'individu, l'on peut trouver des cycles hamiltoniens dans le complémentaire du graphe, c'est-à-dire dans le graphe précisant les compatibilités entre les personnes. Il nous suffira de choisir ensuite le groupe ayant le plus d'individus. Or, un raisonnement de ce type où l'on devra trouver des cycles hamiltoniens dans le complémentaire du graphe pourrait nous donner une configuration possible parmi tant d'autres.

Mais comment s'assurer que l'on a trouvé la solution la plus optimisée ? En d'autres termes, comment prouver que n individus invités s'agit du nombre d'invités maximal sans qu'il y ait de conflit.

Cette solution ne pourrait être efficace seulement dans des graphes particuliers. Cette approche étant limitée, elle n'est pas généralisable à d'autres situations éventuellement plus complexes. Après plusieurs recherches, nous observons que cette approche ne donnera pas forcément la solution la plus optimale, mais seulement une configuration possible. En vue de ses limitations, cette approche ne sera pas implémentée sous forme de code, mais fera l'objet d'une évolution dans la solution finale.

4.3 Approche et solution finale :

La question, en termes de propriétés du graphe, se pose maintenant ainsi : combien de familles de sommets doit-on créer au minimum, si l'on veut que deux sommets liés par une arête n'appartiennent jamais à la même famille ? Nous allons considérer que chaque famille est caractérisée par une couleur, et développer quelques éléments de théorie à ce sujet.

En théorie des graphes, colorer un graphe signifie attribuer une couleur à tous les sommets de telle sorte que deux sommets adjacents soient de couleurs différentes. On appelle nombre chromatique $\chi(G)$ d'un graphe le nombre minimum de couleurs nécessaires pour colorier chaque sommet du graphe de façon que deux sommets adjacents soient de couleurs différentes. Le problème de coloration qui consiste à trouver le nombre chromatique $\chi(G)$ d'un graphe donné est un problème difficile. L'algorithme de *Welsh et Powell* est un algorithme qui se base sur les degrés des sommets pour proposer une assez bonne coloration du graphe. Cependant l'algorithme n'assure pas toujours que la coloration soit minimale, nous allons donc l'améliorer afin de l'adapter à notre situation.

Dans notre cas l'on pourrait chercher le nombre chromatique $\chi(G)$ du graphe, or ce qui nous intéresse ici s'agit du nombre maximal de collègues que l'on pourrait inviter, et non la création d'un minimum de familles de sommets, même si cela pourrait nous donner une réponse approchée.

Le graphe est formé de 11 sommets $A, B, C, D, E, F, G, H, I, J, K$ qui représentent les 11 collègues et d'arêtes qui expriment les incompatibilités d'humeur. Afin de pouvoir déterminer le plus grand nombre d'invités ne rencontrant pas de tensions lors de la soirée, nous allons procéder suivant ce raisonnement :

4.3.1 Vérification d'une chaîne eulérienne

Le problème revient à chercher une chaîne eulérienne. On vérifie aisément que le graphe est connexe (voir graphe). En effet, $H - G - F - E - D - C - B - A - K - I - J$ est une chaîne passant par tous les sommets du graphe. Donc deux sommets quelconques du graphe sont reliés par une chaîne.

4.3.2 Appliquons l'algorithme de Welsh et Powell (optimisé à notre situation) :

| Sommets | H | B | J | C | F | K | D | E | I | G | A |
|-------------------|----|----|----|----|----|----|----|----|----|----|----|
| Degré croissant | 1 | 2 | 2 | 3 | 3 | 3 | 4 | 4 | 4 | 6 | 8 |
| Couleur attribuée | C1 | C1 | C1 | C2 | C1 | C1 | C3 | C2 | C2 | C3 | C4 |

FIGURE 4.1 – Tableau des couleurs

- **Étape 1 :** On range les sommets dans l'ordre croissant de leur degré (voir tableau ci-dessus).
Remarque : Dans l'algorithme original de Welsh-Powell, nous aurions rangé les sommets dans l'ordre décroissant, nous faisons ici l'inverse, car nous avons défini une règle de priorité.
- **Étape 2 :** Bien évidemment, nous trouverons plusieurs configurations possibles, pour remédier à ce souci, il suffit de suivre un ordre. On choisit le sommet ayant le moins de voisins (et donc le moins de liaisons) dans l'ordre du graphe, ici il s'agit de H (Hector) et l'on attribue à ce sommet une couleur (C1).
- **Étape 3 :** En suivant la liste, on attribue cette couleur (C1) à tous les sommets qui ne lui sont pas adjacents et qui ne sont pas adjacents entre-eux. En d'autres termes, il faut trouver les n sommets qui n'ont aucune arête entre eux, bien évidemment dans l'ordre du tableau prédéfini à l'étape 1. Au fil du déroulement de l'algorithme, une règle sera toujours valable et devra être respectée à chaque itération :

Règle

$$\text{Degré du noeud actuel} \geq \text{Degré du noeud précédent}$$

De cette manière, chaque individu sera hiérarchisé de sorte à avantager les individus 'moins difficiles', en d'autres termes, les individus ayant le moins de problèmes d'incompatibilités d'humeur seront choisis en premier et donc mis en valeur. En définissant cette règle, nous optimisons notre résultat et donc le nombre possible de personnes qui sont capables de se supporter entre elles.

- **Étape 4 :** On attribue une deuxième couleur au premier sommet non colorié et on recommence depuis l'étape 2.
- **Étape 5 :** On réitère les étapes précédentes jusqu'à épuisement des sommets.

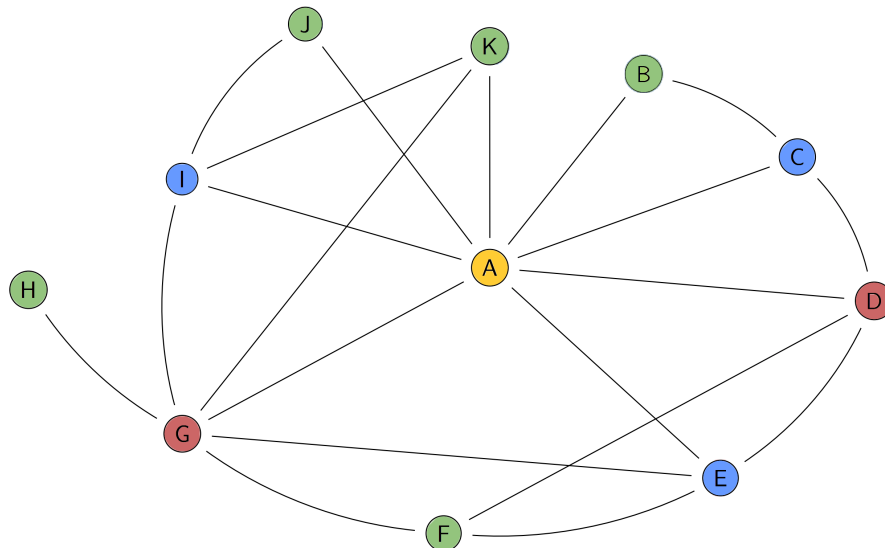


FIGURE 4.2 – Coloration du graphe selon une variante de l'algorithme de Welsh-Powell

- **Étape 6 :** Finalement, nous récupérerons le nombre et les propriétés de tous les sommets colorés avec la couleur C1, car il s'agit du groupe ayant le plus grand nombre de sommets. Christophe invitera donc tous les sommets colorés en vert :

| K | H | B | F | J |
|-------|--------|----------|--------|-------|
| Kevin | Hector | Benjamin | Franck | Jacob |

FIGURE 4.3 – Résultat final du groupe à inviter

5. Programmation informatique en Java

Informatiquement parlant, l'algorithme de *Welsh-Powell* se déroulera via les règles suivantes :

1. Ranger les sommets dans l'ordre croissant des degrés.
2. Tant que tous les sommets ne sont pas colorés :
 - Considérer une couleur *C1* (en *Integer*), différente des couleurs déjà utilisées.
 - Considérer le premier sommet non encore coloré dans l'ordre croissant des degrés et lui affecter la couleur *C1*.
 - Considérer chacun des autres sommets non colorés dans l'ordre croissant des degrés.
 - (a) S'il est adjacent à un sommet déjà coloré en *C1*, ne lui affecter aucune couleur.
 - (b) Sinon, lui attribuer la couleur *C1*.
 - L'algorithme est terminé dès que tous les sommets sont colorés.

Pour résumer, nous appliquerons le schéma simplifié suivant à notre algorithme :

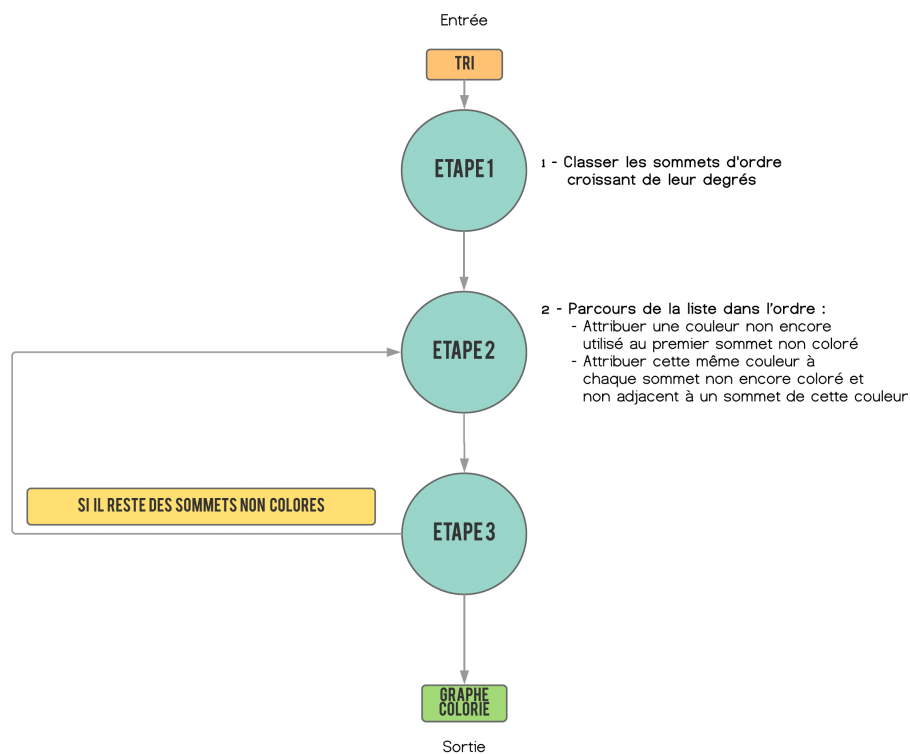


FIGURE 5.1 – Schéma fonctionnel de l'algorithme

Compte tenu de notre problème de départ, nous prendrons soin de convertir chacune des initiales de chaque personne, par son chiffre adéquat (type *Integer*), pour faciliter les manipulations des listes et des mappages : *Map*, *HashMap*, *List*, *ArrayList*. Il est important par ailleurs, de prendre en compte les doublons. Même si cela n'altère pas le résultat final, cela aura pour effet d'augmenter le temps de la simulation finale. Nous prendrons donc soin de ne pas compter les cases interdites(cases

rouges) et les doublons(cases jaunes). Nous implémenterons seulement les cases vertes.

| | A→0 | B→1 | C→2 | D→3 | E→4 | F→5 | G→6 | H→7 | I→8 | J→9 | K→10 |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|
| A→0 | | X | X | X | X | | X | | X | X | X |
| B→1 | X | | X | | | | | | | | |
| C→2 | X | X | | X | | | | | | | |
| D→3 | X | | X | | X | X | | | | | |
| E→4 | X | | | X | | X | X | | | | |
| F→5 | | | | X | X | | X | | | | |
| G→6 | X | | | | X | X | | X | X | | X |
| H→7 | | | | | | | X | | | | |
| I→8 | X | | | | | | X | | | X | X |
| J→9 | X | | | | | | | | X | | |
| K→10 | X | | | | | | X | | X | | |

FIGURE 5.2 – Implémentation de la matrice sous Java

Comme évoqué précédemment, les seules informations prises en compte pour l'implémentation du graphique dans le code Java sans risque de doublons, s'agit des cases vertes du tableau matriciel ¹. La liaison d'un sommet à un autre est faite par un objet *Arete*, de la manière suivante :

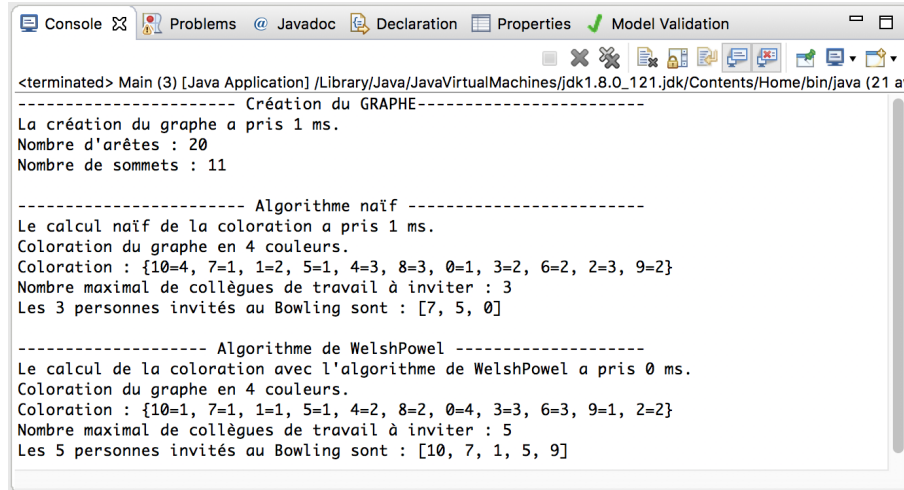
```
Arete A_B = new Arete(this.getSommets().get(0), this.getSommets().get(1));
Arete E_G = new Arete(this.getSommets().get(4), this.getSommets().get(6));
this.arettes.add(A_B);
this.arettes.add(E_G);
```

La 1^{er} couleur C1 utilisés dans le graphe sera la couleur possédant le plus de sommets, car un sommet ayant le moins de liaisons est considéré comme prioritaire. Nous nous baserons donc sur la couleur C1 pour déterminer le nombre maximal de collègues de travail à inviter, ainsi que l'énumération de leurs noms.

1. Informatiquement, la matrice associée à un graphe non orienté à n sommets $S_1, S_2, S_3, \dots, S_n$ est une matrice carrée $M = (a_{ij})_{1 \leq i, j \leq n}$ où le terme a_{ij} est égal à 1 si S_i est adjacent à S_j . Et 0 sinon.

6. Résultat

Voici la simulation informatique du problème résolu manuellement précédemment :



```
<terminated> Main (3) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/bin/java (21 av
----- Création du GRAPHE-----
La création du graphe a pris 1 ms.
Nombre d'arêtes : 20
Nombre de sommets : 11

----- Algorithme naïf -----
Le calcul naïf de la coloration a pris 1 ms.
Coloration du graphe en 4 couleurs.
Coloration : {10=4, 7=1, 1=2, 5=1, 4=3, 8=3, 0=1, 3=2, 6=2, 2=3, 9=2}
Nombre maximal de collègues de travail à inviter : 3
Les 3 personnes invitées au Bowling sont : [7, 5, 0]

----- Algorithme de WelshPowel -----
Le calcul de la coloration avec l'algorithme de WelshPowel a pris 0 ms.
Coloration du graphe en 4 couleurs.
Coloration : {10=1, 7=1, 1=1, 5=1, 4=2, 8=2, 0=4, 3=3, 6=3, 9=1, 2=2}
Nombre maximal de collègues de travail à inviter : 5
Les 5 personnes invitées au Bowling sont : [10, 7, 1, 5, 9]
```

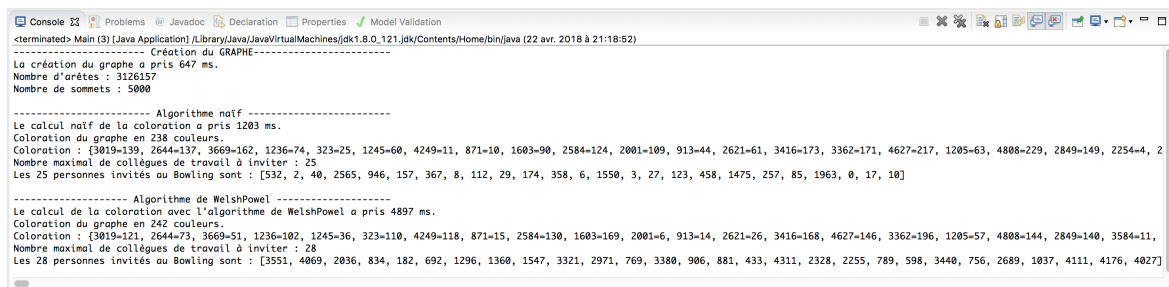
FIGURE 6.1 – Simulation avec un graphe prédéfini à 11 sommets sous eclipse

Nous remarquons très facilement que l'algorithme naïf est rapide, mais donne de mauvais résultats avec un maximum de seulement trois invités. À l'inverse, l'algorithme de *Welsh-Powel* modifié à ma guise est bien plus lent, mais donne un résultat plus efficace avec un maximum de cinq invités. Ici nous retrouvons exactement les cinq invités trouvés lors de la simulation manuelle :

| 10 → K | 7 → H | 1 → B | 5 → F | 9 → J |
|--------|--------|----------|--------|-------|
| Kevin | Hector | Benjamin | Franck | Jacob |

FIGURE 6.2 – Résultat final du groupe à inviter

Afin de pousser nos observations encore plus loin, à des fins plus poussées et des situations réelles : il a été développé dans le programme, la possibilité de créer un graphe avec le nombre de sommets prédéfinis mais avec un poids aléatoire entre chaque sommet. Générant ainsi un graphe lié de manière purement aléatoire selon un seuil de poids prédéfini au départ. Cela permettra de simuler des cas et des situations réelles.



```
<terminated> Main (3) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/Home/bin/java (22 avr. 2018 à 21:18:52)
----- Création du GRAPHE-----
La création du graphe a pris 647 ms.
Nombre d'arêtes : 3126157
Nombre de sommets : 5000

----- Algorithme naïf -----
Le calcul naïf de la coloration a pris 1203 ms.
Coloration du graphe en 238 couleurs.
Coloration : {3819=139, 2644=137, 3669=162, 1236=74, 323=25, 1245=60, 4249=11, 871=10, 1603=90, 2584=124, 2001=109, 913=44, 2621=61, 3416=173, 3362=171, 4627=217, 1205=63, 4808=229, 2849=149, 2254=4, 2
Nombre maximal de collègues de travail à inviter : 25
Les 25 personnes invitées au Bowling sont : [532, 2, 40, 2565, 946, 157, 367, 8, 112, 29, 174, 358, 6, 1550, 3, 27, 123, 458, 1475, 257, 85, 1963, 0, 17, 10]

----- Algorithme de WelshPowel -----
Le calcul de la coloration avec l'algorithme de WelshPowel a pris 4897 ms.
Coloration du graphe en 242 couleurs.
Coloration : {3819=121, 2644=73, 3669=51, 1236=102, 1245=36, 323=110, 4249=118, 871=15, 2584=130, 1603=169, 2001=6, 913=14, 2621=26, 3416=168, 4627=146, 3362=196, 1205=57, 4808=144, 2849=140, 3584=11,
Nombre maximal de collègues de travail à inviter : 28
Les 28 personnes invitées au Bowling sont : [3551, 4069, 2036, 834, 182, 692, 1296, 1360, 1547, 3321, 2971, 769, 3380, 906, 881, 433, 4311, 2328, 2255, 789, 598, 3440, 756, 2689, 1037, 4111, 4176, 4027]
```

FIGURE 6.3 – Simulation avec un graphe aléatoire à 5000 sommets sous eclipse

7. Performance

Plus un algorithme est efficace (nombre de couleurs minimal et nombre maximal de collègues à inviter) et plus cet algorithme est lent. À l'inverse, plus celui-ci est rapide et moins ses résultats sont optimaux. Ainsi l'algorithme naïf est rapide, mais donnera assez rapidement de mauvais résultats (un nombre de couleurs éloigné de l'optimale). Cependant l'algorithme de *Welsh-Powel* modifié est le plus lent, mais le plus efficace. L'algorithme de *Welsh-Powel* d'origine est quant à lui un bon compromis, il offre un temps de calcul très proche de celui de l'algorithme naïf tout en gardant une précision semblable à l'algorithme final. On constate assez vite les différences à partir d'environ 100 sommets tout en considérant les poids des arêtes. Les valeurs numériques de certaines expériences réalisées sont disponibles en commentaire de la classe *Main* du projet (cf. Annexe).

8. Conclusion générale

Conclusion

Ce TP m'a permis d'avoir une approche plus concrète et plus approfondie des notions théoriques abordées en cours. Le temps consacré pour cette recherche a été très enrichissant, sur l'appréhension de la notion de la recherche opérationnelle et de son optimisation.

De manière générale, il s'agissait d'une première approche, permettant de mieux comprendre et maîtriser les notions de base importantes de théorie des graphes, mais aussi d'acquérir une expérience sur le fait qu'un problème rencontré permet d'être surmonté par l'intermédiaire de différentes techniques de graphe selon plusieurs critères.

Sitographie

- Introduction à la théorie des graphes - Didier Müller :
<http://www.nymphomath.ch/graphes/graphes.pdf>
- Théorie des Graphes - Jean-Charles Régin, Arnaud Malapert :
http://www.i3s.unice.fr/~malapert/org/teaching/graphes/slza63_cours.pdf
- Optimisation linéaire en nombres entiers :
https://fr.wikipedia.org/wiki/Optimisation_linéaire_en_nombres_entiers
- Recherche Opérationnelle PL - Benoît Darties :
https://cours.infotro.fr/wp-content/uploads/CM_PL.pdf
- Forum mathématique :
<http://mathematiques.ac.free.fr/>
- Coloration de graphes :
http://www.gymomath.ch/javmath/polycopie/th_graphe7.pdf
- Algorithmes et modélisation de graphes - Olivier Briant :
http://caseine.org/pluginfile.php/7556/mod_resource/content/1/graphes.pdf
- Cours sur l'optimisation des graphes :
<http://www.academie-en-ligne.fr/Ressources/7/MA04/AL7MA04TEPA0013-Sequence-03.pdf>
- Programmation linéaire en nombres entiers :
<http://www.iecl.univ-lorraine.fr/~Jean-Francois.Scheid/Enseignement/plne.pdf>

9. Annexe

Code source : Algorithme 'naïf'

```
import java.util.LinkedList;
import java.util.List;
import entite.AbstractGraphe;
import entite.Sommet;
/** Classe modélisant la coloration selon un algorithme naïf de départ.
 * @author Vincent CANDAPPANE*/
public class Naif extends AbstractColoration{

    public Naif(AbstractGraphe g){
        /** Créé une nouvelle instance pour une colorisation de graphe.
         * @param g le graphe à colorer. */
        super(g);
    }

    public void execute(){
        // Initialisation des variables
        List<Integer> couleurChoisie = new LinkedList<Integer>();
        int j;
        int size = this.liste_sommets.size();

        // On parcourt tout les sommets du graphe
        for(int i = 0 ; i < size ; i++){
            couleurChoisie.clear();

            // On cherche les couleurs de ses voisins
            for(Sommet s : this.liste_sommets.get(i).getVoisins()){
                if(!affectation_couleurs.containsKey(s))
                    continue;
                if(couleurChoisie.contains(affectation_couleurs.get(s)))
                    continue;

                couleurChoisie.add(affectation_couleurs.get(s));
            }

            j = 1;
            // On attribue au sommet courant la plus petite couleur non trouvée
            //   chez ses voisins
            while(couleurChoisie.contains(j)){
                ++j;
            }
            affectation_couleurs.put(this.liste_sommets.get(i), j);
        }

        for(Integer i : affectation_couleurs.values())
            if(i > this.nb_couleurs)
                this.nb_couleurs = i;
    }
}
```


Code source : Algorithme de 'Welsh et Powell'

```

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import utils.DegreSommetComparator;
import entite.AbstractGraphe;
import entite.Sommet;
/**
 * Classe modélisant la coloration selon l'algorithme de Welsh-Powell.
 * @author Vincent CANDAPPANE
 */
public class WelshPowel extends AbstractColoration {
    public WelshPowel(AbstractGraphe g) {
        /** Créé une nouvelle instance pour une colorisation selon Welsh-Powell d'un graphe.
         * @param g le graphe à colorer. */
        super(g);
    }
    public void execute() {
        List<Sommet> sommets = new ArrayList<Sommet>(this.liste_sommets);
        // Reinitialisation de la variable (en cas de réutilisation).
        this.affectation_couleurs = new HashMap<Sommet,Integer>();
        this.nb_couleurs = 0;
        // Tri la liste des sommets
        Collections.sort(sommets,new DegreSommetComparator());
        // On prend les sommets dans l'ordre croissant.
        while (!sommets.isEmpty()) {
            Sommet s = sommets.get(0);
            this.nb_couleurs++;
            // On met la plus petite couleur
            this.affectation_couleurs.put(s, this.nb_couleurs);
            sommets.remove(s);
            List<Sommet> copy = new ArrayList<Sommet>(sommets);
            // On parcourt les autres sommets pour attribuer la couleur ailleurs
            for (Sommet s2 : sommets) {
                // On passe les sommets ayant déjà été colorés
                if (!this.affectation_couleurs.containsKey(s2)) {
                    if (!this.aUnVoisinColoreAvec(s2,this.nb_couleurs)) {
                        // Si non-adjacent, on attribue la couleur
                        this.affectation_couleurs.put(s2, nb_couleurs);
                        copy.remove(s2);
                    }
                }
            }
            sommets = copy;
        }
    }
    public boolean aUnVoisinColoreAvec(Sommet sommet, int couleurCode) {
        //Tester si le sommet a un voisin coloré avec une couleur passée en paramètre.
        for (Sommet s : sommet.getVoisins()) {
            // Des qu'on trouve la couleur on renvoie 'true'.
            if (this.affectation_couleurs.containsKey(s) &&
                ↪ this.affectation_couleurs.get(s) == couleurCode)
                return true;
        }
        return false;
    }
}

```

Code source : Récupération d'informations sur le graphe

```
public abstract class AbstractColoration {
    protected AbstractGraphe g;
    protected int nb_couleurs;
    protected Map<Sommet, Integer> affectation_couleurs;
    protected List<Sommet> liste_sommets;

    public AbstractColoration(AbstractGraphe g){
        this.g = g;
        this.nb_couleurs = 1;
        this.liste_sommets = this.g.getSommets();
        this.affectation_couleurs = new HashMap<Sommet, Integer>();
    }

    public abstract void execute();
    public int getNbCouleurs(){
        return this.nb_couleurs;
    }

    public Map<Sommet, Integer> getColoration(){
        return this.affectation_couleurs;
    }

    public int getMaximumInvitation(){
        int count = Collections.frequency(new
            ↪ ArrayList<Integer>(affectation_couleurs.values(), 1);
        return count;
    }

    public Collection<Integer> getInvitationName() {
        Collection<Integer> result = new ArrayList<Integer>();
        for(Sommet s : affectation_couleurs.keySet()) {
            if((affectation_couleurs.get(s) == 1))
                result.add(s.getID());
        }
        return result;
    }
}
```

Code source : Main

```
import algo.AbstractColoration;
import algo.Naif;
import algo.WelshPowel;
import entite.AbstractGraphe;
import entite.GrapheAleatoire;
import entite.GraphePredefini;

public class Main {
    public static void main(String[] args) {
        long deb, fin;
        int nb_couleurs;
        AbstractColoration coloration;
        deb = System.currentTimeMillis();

        // ----- Création d'un graphe prédéfini manuellement -----
        AbstractGraphe g = new GraphePredefini(11);
        fin = System.currentTimeMillis() - deb;
        System.out.println("La création du graphe a pris " + fin + " ms.");
        System.out.println(g);
    }
}
```

```

// ----- Algorithme naïf -----
coloration = new Naif(g);
deb = System.currentTimeMillis();
coloration.execute();
nb_couleurs = coloration.getNbCouleurs();
fin = System.currentTimeMillis() - deb;
System.out.println("----- Algorithme naïf -----");
System.out.println("Le calcul naïf de la coloration a pris " + fin + "ms.");
System.out.println("Coloration du graphe en " + nb_couleurs + "couleurs.");
System.out.println("Coloration : " + coloration.getColoration());
System.out.println("Nombre maximal de collègues de travail à inviter : " +
    ↳ coloration.getMaximumInvitation());
System.out.println("Les " + coloration.getMaximumInvitation() + " personnes
    ↳ invités au Bowling sont : " + coloration.getInvitationName());
System.out.println();

// ----- Algorithme de WelshPowel -----
coloration = new WelshPowel(g);
deb = System.currentTimeMillis();
coloration.execute();
nb_couleurs = coloration.getNbCouleurs();
fin = System.currentTimeMillis() - deb;
System.out.println("----- Algorithme de WelshPowel -----");
System.out.println("Le calcul de la coloration a pris " + fin + " ms.");
System.out.println("Coloration du graphe en " + nb_couleurs + "couleurs.");
System.out.println("Coloration : " + coloration.getColoration());
System.out.println("Nombre maximal de collègues de travail à inviter : " +
    ↳ coloration.getMaximumInvitation());
System.out.println("Les " + coloration.getMaximumInvitation() + " personnes
    ↳ invités au Bowling sont : " + coloration.getInvitationName());
}

}

/*
 * Simulations et Observations :
 *
 *      Graphe(N=50,E=368) :
 *          - Naïf :                  10 couleurs pour 2ms
 *          - Welsh-Powel :          8 couleurs pour 4ms
 *
 *      Graphe(N=50,E=611) :
 *          - Naïf :                  13 couleurs pour 1ms
 *          - Welsh-Powel :          11 couleurs pour 2ms
 *
 *      Graphe(N=100,E=1452) :
 *          - Naïf :                  14 couleurs pour 2ms
 *          - Welsh-Powel :          13 couleurs pour 5ms
 *
 *      Graphe(N=200,E=5884) :
 *          - Naïf :                  22 couleurs pour 8ms
 *          - Welsh-Powel :          21 couleurs pour 8ms
 *
 *      Graphe(N=200,E=9980) :
 *          - Naïf :                  37 couleurs pour 10ms
 *          - Welsh-Powel :          34 couleurs pour 15ms
 *
 *      Graphe(N=1000,E=250220) :
 *          - Naïf :                  126 couleurs pour 52ms
 *          - Welsh-Powel :          123 couleurs pour 647ms
 *
 *      Graphe(N=5000,E=3126157) :
 *          - Naïf :                  238 couleurs pour 1203ms
 *          - Welsh-Powel :          242 couleurs pour 4897ms
 */

```