Rapport de projet : Coloration de graphes appliquées au problème d'emplois du temps.

Optimisation dans les graphes

Matthieu Speismann
Guillaume Tran-Ruesche
Théo Guella

Sommaire

I. Présentation du problème

a) Situation pratique

Pour tout établissement scolaire ou d'enseignement supérieur, l'un des principaux défis à relever avant la rentrée des classes est la mise au point des emplois du temps des différents élèves en fonction des cours et options que chacun doit suivre. Dans ce problème, plusieurs facteurs sont à prendre en compte :

- Le choix de cours des élèves, chaque élève a la possibilité de choisir ses propres cours et a par conséquent son propre set de cours.
- La disponibilité des professeurs selon les créneaux de cours. En effet, les professeurs ayant une vie personnelle et pouvant enseigner dans d'autres établissement, ils ne sont pas disponibles sur l'ensemble des créneaux de cours de l'établissement.
- Le nombre de créneaux disponibles sur l'ensemble de la semaine est limité.
- Le nombre de salles de cours de l'établissement est aussi limité.
- La durée des différents cours peut être variable. En effet, suivant les matières enseignées, il faut allouer des plages horaires plus ou moins grandes à chaque cours. Nous avons pris la décision dans ce rapport d'utiliser des cours de type universitaire avec des plages faites par demi-journée et de durée identique.

L'objectif principal des établissements est d'obtenir un emploi du temps viable, pas obligatoirement optimal, qui respecte l'ensemble des critères précédents.

b) Modèle de la situation

La situation décrite dans la partie précédente peut être modélisée par la coloration d'un graphe d'incompatibilités entre les cours et dont les couleurs représentent les différents créneaux horaires possibles.

La construction du graphe est la suivante :

- Les nœuds représentent les différents cours dispensés par l'établissement.
- Si un même élève suit deux cours, alors ces deux cours sont incompatibles car l'élève ne peut pas suivre les deux cours sur le même créneau. On ajoute une nouvelle arrête entre les nœuds représentants ces deux cours.
- De même, si un professeur enseigne 2 cours, on ajoute une nouvelle arrête entre les nœuds représentants ces deux cours.

On cherche ensuite une coloration de ce graphe (une couleur correspondant à un créneau précis) en testant à chaque fois les disponibilités des professeurs sur le créneau concerné et le nombre de salles de cours utilisées sur ce créneau, soit le nombre de fois où cette couleur a été donné.

c) Modèle mathématique

Sets :

- $Elèves$
- $Professeurs$
- $Cours$
- $Créneaux$ (= Couleurs dans la manipulation du graphe)

Paramètres :

- $DisponibilitéProfesseurs_{i,k} : Professeurs \times Créneaux$
- $CoursSuivisElèves_{i,j} : Cours \times Elèves$
- $CoursDonnésProfesseurs_{i,j} : Cours \times Professeurs$
- $NombreSalles$

Des paramètres précédents, on peut déduire les deux paramètres suivants :

- $DisponibilitésCours_{i,k} : Cours \times Créneaux$, 1 si le cours peut avoir lieu sur le créneau k, 0 sinon
- $Incompatibilités_{i,j} : Cours \times Cours$, 1 si les cours i et j sont incompatibles, 0 sinon

Variable :

$Coloration_{i,k} : Cours \times Créneaux$, 1 si le cours i est associé au créneau k, 0 sinon

Objectif : On minimise le créneau maximal utilisé, si celui-ci est bien inférieur au nombre de créneau, alors la coloration est viable.

$$\min(\max(\{k \setminus (\exists\, i \in Cours \setminus X_{i,k} = 1), k \in Créneaux\})$$

Contraintes :

- Assure que si il existe une arrête entre les cours i et j, les deux nœuds ne peuvent pas être sur le même créneau :

$$Incompatibilité_{i,j} * (Coloration_{i,k} + Coloration_{j,k}) \leq 1, \forall\, (i,j) \in Cours^2, \forall\, k \in Créneaux$$

- Vérifie qu'un cours i peut bien avoir lieu sur le créneau k :

$$Coloration_{i,k} \leq DisponibilitéCours_{\mathrm{i,k}}, \forall\, (i,k) \in Cours \times Créneaux$$

- Respect du nombre de salles :

$$\sum_{i \in Cours} Coloration_{i,k} \leq Nombre\_salles$$

- Un créneau par cours :

$$\sum_{k \in Créneaux} Coloration_{i,k} = 1, \forall\, i \in Cours$$

II. Génération des données pour la simulation

a) Intérêt et modélisation

Notre projet visant à illustrer la création d'un emploi du temps dans un collège, nous avons décidé de créer un fichier permettant de générer aléatoirement, selon des paramètres que l'utilisateur peut modifier, une base de données entre des élèves, des professeurs et des matières. Pour établir les liens entre ces trois entités, nous avons décidé de faire appel au hasard. Pour chaque élève, chaque matière lui sera associée (ou non) avec une certaine probabilité. De même, chaque matière se verra associée à un professeur (et un seul) de façon aléatoire. Si un élève n'a aucun cours associé à la fin du parcours des différents cours, alors on lui en associe un aléatoirement. Un professeur peut gérer plusieurs cours, mais il peut également n'en générer aucun. Dans tous les cas, un professeur qui ne donne pas de cours n'apparaîtra pas dans la suite du projet.

Nous avons fait ce choix de modélisation, car nous ne souhaitions pas travailler avec une configuration fixe (par exemple, 5 élèves, 2 professeurs, 3 matières, l'élève 1 suit les cours 1 et 2, l'élève 2 suit seulement le cours 3, etc.). Dans notre cas, nous avons la possibilité de tester nos programmes sur plusieurs situations d'une même configuration (c'est-à-dire avec des paramètres égaux), mais également en faisant varier un ou plusieurs paramètres. Cela nous donne la possibilité de voir plus facilement certains problèmes, de créer et tester nos programmes sur une petite base de données pour ensuite les utiliser sur une base de données à échelle réelle, et également de voir les limites de nos algorithmes (c'est-à-dire quelles configurations ne donnent pas de solutions ? À cause de quels paramètres ?). Le choix d'attribuer un professeur à une matière de façon aléatoire a également été effectué pour complexifier notre modélisation. En effet, si chaque cours avait un seul professeur et un professeur avait un unique cours, le rôle du « professeur » aurait ici été inutile (car confondu avec le rôle de la matière). En donnant la possibilité à un enseignant de donner plusieurs cours, cela ajoute des incompatibilités (en effet, il ne pourra pas donner ses cours en même temps, même si les deux groupes d'élèves sont différents) et rend notre projet plus intéressant (même s'il y a moins de solutions réalisables).

Enfin, toujours dans l'objectif de modéliser une situation réelle, et de ne pas se perdre entre des numéros, des lettres ou des caractères spéciaux, nous avons souhaité donner une identité à chaque élément généré par la simulation. Pour cela, nous avons récupéré des listes de données : une liste de 200 prénoms (Gabriel, Léa, etc.) présente dans le fichier `prenoms.txt`, une liste de 196 noms (Petit, Durand, etc.) présentes dans `noms.txt`, et une liste de 22 matières (Français, Mathématiques, etc.) dans `cours.txt`.

b) Présentation du code

Tout d'abord, il est nécessaire que l'on définisse les paramètres principaux :

- `studentNumber`, `courseNumber`, `profsNumber` : Ces paramètres permettent de définir respectivement combien d'étudiants, de matières et de professeurs différents on souhaite avoir dans notre collège virtuel. Tous ces paramètres sont des entiers variants entre 0 et $\mathbb{N}$.
- `courseStudentProbability` : Ce paramètre permet de définir avec quelle probabilité on associe un cours à un élève. On le prendra souvent faible pour avoir des solutions réalisables. Ce paramètre est un réel variant entre 0 et 1 (exclus)
- `availableDays`, `availabilityProbability` : Ces deux paramètres permettent de générer la disponibilité des professeurs. Le premier fixe le nombre de jours sur lequel on veut faire notre emploi du temps, et le deuxième définit avec quelle probabilité (souvent élevée pour avoir des solutions réalisables) un professeur sera disponible chaque jour. `availableDays` est un entier naturel et `availabilityProbability` est une probabilitée.

Une fois ces paramètres définis, nous créons une matrice de taille `profsNumber` x `availableDays` avec pour chaque coefficient une valeur de 1 avec une probabilité `availabilityProbability`, et une valeur de 0 sinon. Ainsi, pour la ligne `i` et la colonne `j`, cela signifie que le professeur `i` est disponible le jour `j` si on a un 1, et qu'il ne l'est pas si on a un 0. On vient ensuite lire les trois documents .txt et choisir pour chacun `n` mots (où n correspond à `studentNumber`, `courseNumber` ou `profsNumber` selon la liste) afin de créer notre trois listes (élèves, professeurs et matières)

La partie d'initialisation étant définie, nous créons ensuite notre liste des arêtes élèves/cours `courseStudentEdges` et cours/profs `courseProfEdges`. Comme expliqué précédemment, pour chaque élève on parcourt les matières, et nous créons un lien entre les deux avec une probabilité définie en paramètres. On a également un booléen `hasCourse` défini pour chaque élève qui permet de savoir si un élève n'a pas de cours à la fin de l'itération. Si un élève est dans ce cas-là, alors on lui associera un cours au hasard pour qu'il ne soit pas relié à rien. Pour l'autre partie du graphe, pour chaque matière, on lui associe un professeur au hasard. Cela signifie qu'il est fort probable que des professeurs se retrouvent isolés et ne servent donc à rien dans notre configuration. Cependant, c'est le meilleur moyen que nous avons trouvé pour créer des liens entre nos entités. Ainsi, on ne définit pas vraiment le nombre de professeur, mais illustre plutôt le nombre de matières qu'aura un professeur. En effet, si on a `courseNumber` < `profsNumber`, cela signifie qu'on a peu de matières à attribuer à beaucoup de professeurs. Chaque professeur aura donc peu de matières. Dans le cas contraire, on a beaucoup de matières à associer à peu de professeurs. Ainsi chaque professeur aura beaucoup de cours associés. Ce paramètre doit donc être fixé connaissant `courseNumber`.

Enfin, nous avons tracé un graphe grâce à la bibliothèque `networkX`. Nous avons choisi pour représenter notre configuration, de tracer deux graphes bipartis avec une variable

en commun. En effet, nous affichons un premier graphe biparti entre les élèves (bleu) et les matières (vert), puis un autre entre les matières (les mêmes que précédemment, donc de couleur vert) et les professeurs (rouge). Les deux figures ci-dessous illustrent la génération d'une configuration à travers deux exemples : le premier avec 10 élèves, 6 cours et 5 professeurs, et le deuxième avec 50 élèves, 20 cours et 25 professeurs.

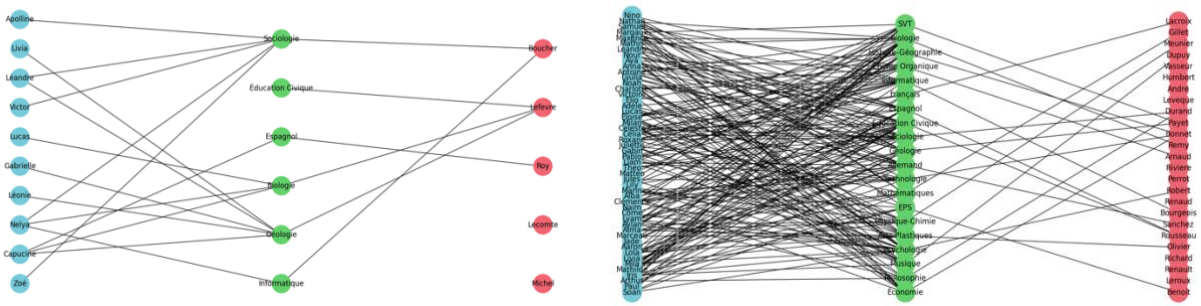

Figure 1 : Premier exemple (gauche) et deuxième exemple (droite)

III. Algorithmes

a) Explication

Pour résoudre notre problème de coloration, nous avons commencé par implémenter des algorithmes pour résoudre le problème sans contraintes supplémentaires (Nombre de salles et disponibilité des professeurs). Pour cela, il existe deux algorithmes, dont l'un, trivial consiste à attribuer à chaque nœud une couleur différente :

Pseudo-code de l'algorithme trivial de coloration :

```
Node_colors := []
For each node in graph do
        For each index_color do
                If color[index] not in node_colors do
                        node_colors.append(color[index])
                        break For
                Else do pass
        End For
End For
Return node_colors
```

La complexité de cet algorithme est en $O(C \times N)$ où C est le nombre de couleurs (et donc de créneaux ouverts) et N le nombre de nœuds (et donc de cours).

Il existe aussi un algorithme glouton dont l'objectif est de parcourir chaque nœud, puis, pour chacun d'eux, d'établir la liste des couleurs déjà attribuées à ses voisins et on choisit d'attribuer à ce nœud la première couleur non présente dans cette liste.

Pseudo-code de l'algorithme glouton de coloration :

```
Node_colors := []
For each node in graph do
        Voisin_colors := []
        For each voisin of node do
                voisin_colors.append(Node_color(voisin))
        End For
        For color in list_colors do
                If color not in voisin_colors do
                        Node_colors.append(color)
                        Break For
Return Node_colors
```

La complexité de cet algorithme est en $O\big(N(N + C \times N)\big) = O\big(N^2(1 + C)\big) = O(C \times N^2)$

Cette méthode est performante mais ne renvoie pas nécessairement le résultat optimal (qui n'est possible que par méthode par force brute, de complexité déraisonnable). On peut s'approcher cependant du résultat optimal avec une amélioration sur l'ordre de parcours des nœuds. En effet, si on parcourt les nœuds dans l'ordre décroissant de leur degré, lors d'un problème standard, on utilise moins de couleur qu'avec l'algorithme glouton de base.

A partir de ces algorithmes standards, on va pouvoir préciser nos conditions supplémentaires :

- Tout d'abord, la condition sur les disponibilités des professeurs. On donne désormais en argument des fonctions de coloration la matrice de disponibilité des cours (obtenu par la disponibilité des professeurs donnant ces cours) et avant d'ajouter une couleur (unique pour chaque créneau de cours), on vérifie que le cours peut effectivement avoir lieu à ce créneau.
- Ensuite, on utilise également la condition sur le nombre de salles de cours disponibles. Avant d'ajouter une couleur à un cours, on va vérifier que le nombre de cours qui ont déjà cette couleur est strictement inférieur au nombre de salle.
  Cette condition ne s'applique que sur les algorithmes gloutons de coloration. En effet, tous les cours ayant une couleur différente pour la coloration triviale, il n'est pas nécessaire d'effectuer cette vérification.

On obtient alors les pseudos-codes suivant :

Pseudo-code personnalisé de l'algorithme trivial de coloration :

```
Node_colors := []
For each node in graph do
        For each index_color do
                If color[index] not in node_colors do
                        If availability_courses[node][index] = 1 do
                                node_colors.append(color[index])
                                break For
                        else do pass
                Else do pass
        End For
End For
Return node_colors
```

Pseudo-code personnalisé de l'algorithme glouton de coloration :

```
Node_colors := []
For each node in graph do
        Voisin_colors := []
        For each voisin of node do
                voisin_colors.append(Node_color(voisin))
        End For
        For color in list_colors do
                If color not in voisin_colors do
                        If availability_courses[node][index] = 1 do
                                If Number_courses(color, Node_colors) < n_rooms
                                Node_colors.append(color)
                                Break For
Return Node_colors
```

Avec :

- availability_courses la matrice binaire des disponibilités des cours (node) en fonction du créneau (index)
- Number_courses(color, Node_colors) la fonction qui renvoie le nombre de cours dans Node_colors qui ont déjà la couleur color. Ces résultat est obtenu par simple parcours.
- n_rooms le nombre de salles disponibles.

b) Tests :

Tout d'abord, nous avons voulu tester si, avec nos contraintes supplémentaires, l'algorithme glouton amélioré optimisait mieux que l'algorithme glouton standard. Il se trouve que lors de nos tests préliminaires, nous avons vérifié que non, l'algorithme amélioré n'apporte pas d'amélioration sur la coloration minimale.

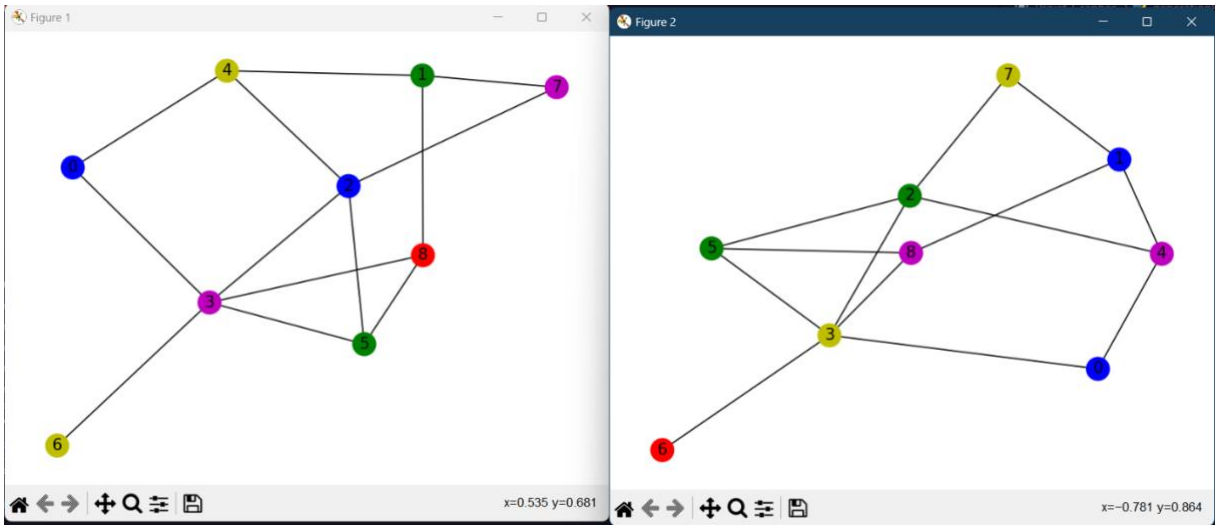

Figure 2 : Comparaison Algorithme Glouton Standard (gauche) et Amélioré (droite)

On constate que la coloration est différente mais le nombre de couleur utilisé étant le même, cet algorithme n'apporte pas une meilleure performance, c'est pour cela que par la suite, pour résoudre notre problème spécifique, nous utiliserons seulement l'algorithme standard.

On veut ensuite vérifier que notre algorithme vérifie l'ensemble des contraintes spécifiques de notre problème. Tout d'abord, on veut vérifier que l'algorithme nous donne une coloration dont les incompatibilités sont effectivement celles prévues par le modèle. Voici alors un exemple obtenu avec les paramètres suivants :

- Nombre d'élève = 5
- Nombre de cours = 5
- Nombre de Prof = 5
- Probabilité de choisir un cours = 0.3
- Nombre de créneau = 4
- Disponibilité des professeurs = 1
- Nombre de salle de classe = 2

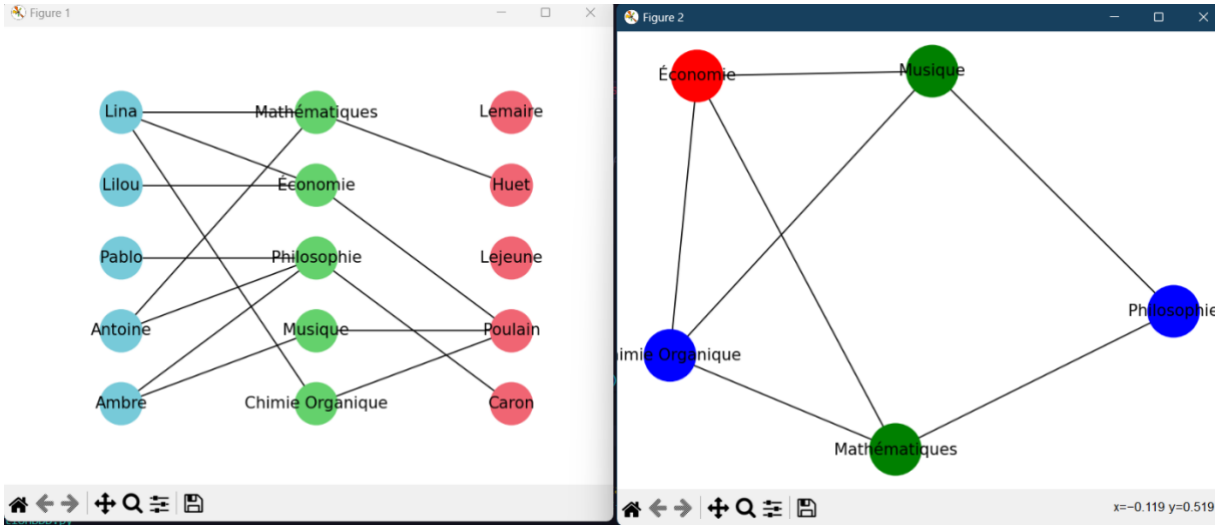

Figure 3 : Graphe représentant la situation (gauche) et graphe d'incompatibilité coloré correspondant (droite)

On confirme que le graphe tracé correspond effectivement aux incompatibilités prévues par l'instance (que ce soit par les élèves ou par les professeurs). On vérifie aussi que la coloration est acceptable, c'est-à-dire que pour chaque nœud, aucun n'a la même couleur qu'un de ses voisins. On vérifie aussi avec cet exemple que le nombre de cours par couleur n'excède par le nombre de salle défini en paramètre (ici 2).

La seconde condition importante à vérifier est la disponibilité des professeurs. On reprend les paramètres précédents, en modifiant seulement la disponibilité des professeurs à 0.7.

On obtient alors la matrice des professeurs suivante :

[[1 0 1 1 1 1 1 0 1 1] *(Chevalier)*
[1 1 1 1 0 1 1 1 0 1] *(Perez)*
[1 0 0 1 1 1 1 0 1 1] *(Leger)*
[1 0 1 1 1 1 1 1 1 0] *(Bernard)*
[0 1 1 1 1 0 0 0 1 1]] *(Lejeune)*

Ce qui correspond à la matrice des cours suivante :

[[1 1 1 1 0 1 1 1 0 1] *(Sociologie)*
[0 1 1 1 1 0 0 0 1 1] *(Géologie)*
[1 1 1 1 0 1 1 1 0 1] *(Economie)*
[1 0 1 1 1 1 1 0 1 1] *(Littérature)*
[1 1 1 1 0 1 1 1 0 1]] *(Psychologie)*

Ce qui correspond effectivement à l'attribution des professeurs aux cours d'après la représentation suivante :

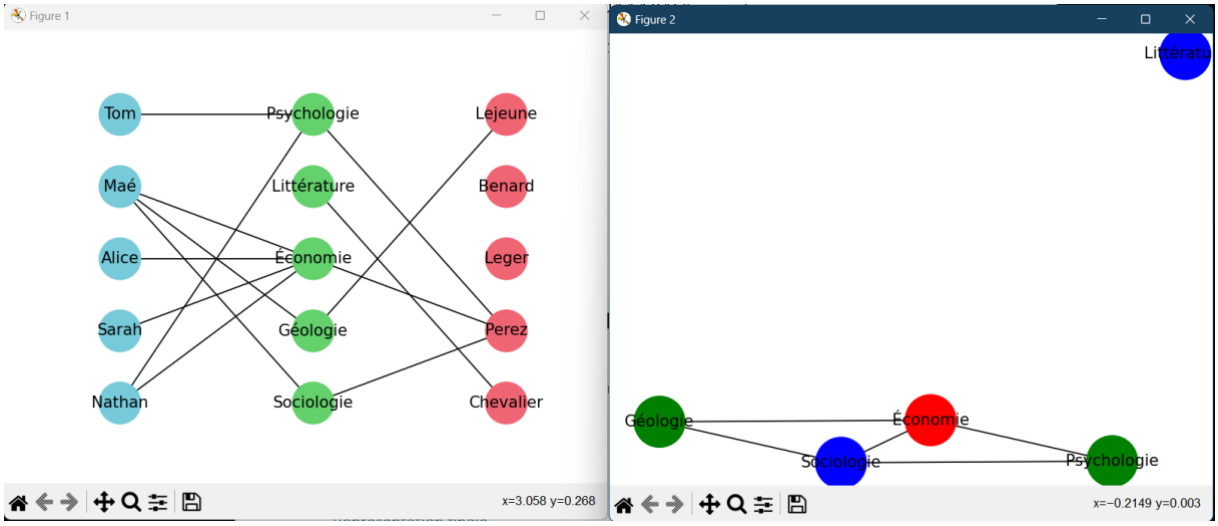

Figure 4 : Graphe représentant la 2e situation (gauche) et graphe d'incompatibilité coloré correspondant (droite)

De plus, les indices des couleurs étant les suivants :

- 0 : Bleu
- 1 : Vert
- 2 : Rouge

On peut confirmer que le cours de sociologie et de littérature peuvent avoir lieu sur le créneau d'indice 0 (d'où sa couleur bleu), que la psychologie et la géologie peuvent être vert (d'indice 1) et que l'économie peut être sur le créneau 2 représenté par le rouge. Ici aussi, la condition sur le nombre de salle est vérifiée.

On veut maintenant essayer la capacité de nos algorithmes à traiter de grandes instances. Voici les paramètres pour cet essai :

- Nombre d'élève = 50
- Nombre de cours = 15
- Nombre de Prof = 12
- Probabilité de choisir un cours = 0.25
- Nombre de créneau = 10 *(2 créneaux par jour)*
- Disponibilité des professeurs = 0.9
- Nombre de salles de classe = 5

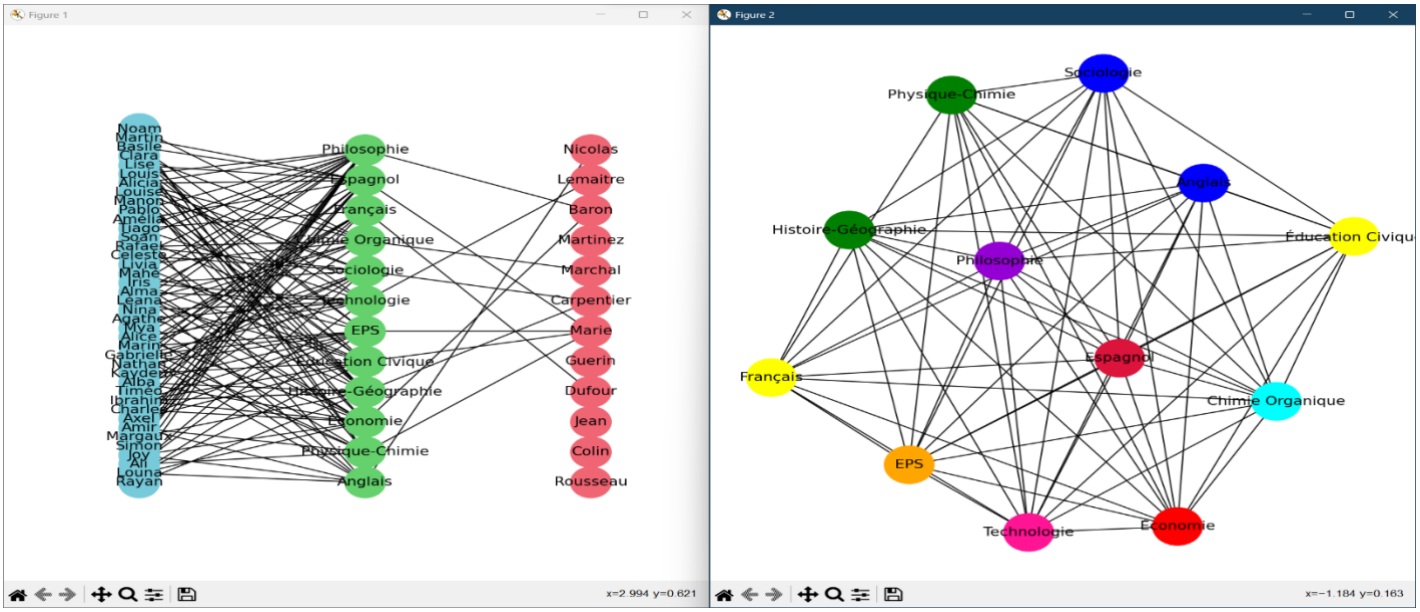

Figure 5 : Graphe représentant la grande instance (gauche) et graphe d'incompatibilité coloré correspondant (droite)

c) Représentation finale :

Une fois notre coloration effectuée, nous avons souhaité réorganiser visuellement le résultat dans le cadre de notre thématique de création d'emplois du temps. Pour cela, nous avons récupéré dans un dictionnaire tous les nœuds (et donc le nom des matières) regroupés par leur couleur (la clé du dictionnaire). Nous avons également créé un tableau semblable à un emploi du temps, avec en colonne les jours de la semaine (de lundi à vendredi, en boucle si jamais il y a plus de 5 colonnes) et en ligne le « matin » et l'« après-midi » pour représenter deux possibilités de créneaux. Nous avons ensuite représenté chaque groupe de matières dans les créneaux, pour voir plus facilement comment on aurait pu organiser un emploi du temps avec notre configuration initiale.

Pour tester cette fonctionnalité, on a essayé avec les paramètres suivants :

- Nombre d'élève = 50
- Nombre de cours = 15
- Nombre de Prof = 12
- Probabilité de choisir un cours = 0.25
- Nombre de créneau = 10 *(2 créneaux par jour)*
- Disponibilité des professeurs = 0.9
- Nombre de salles de classe = 5

Voici la représentation des données et du graphe de coloration :

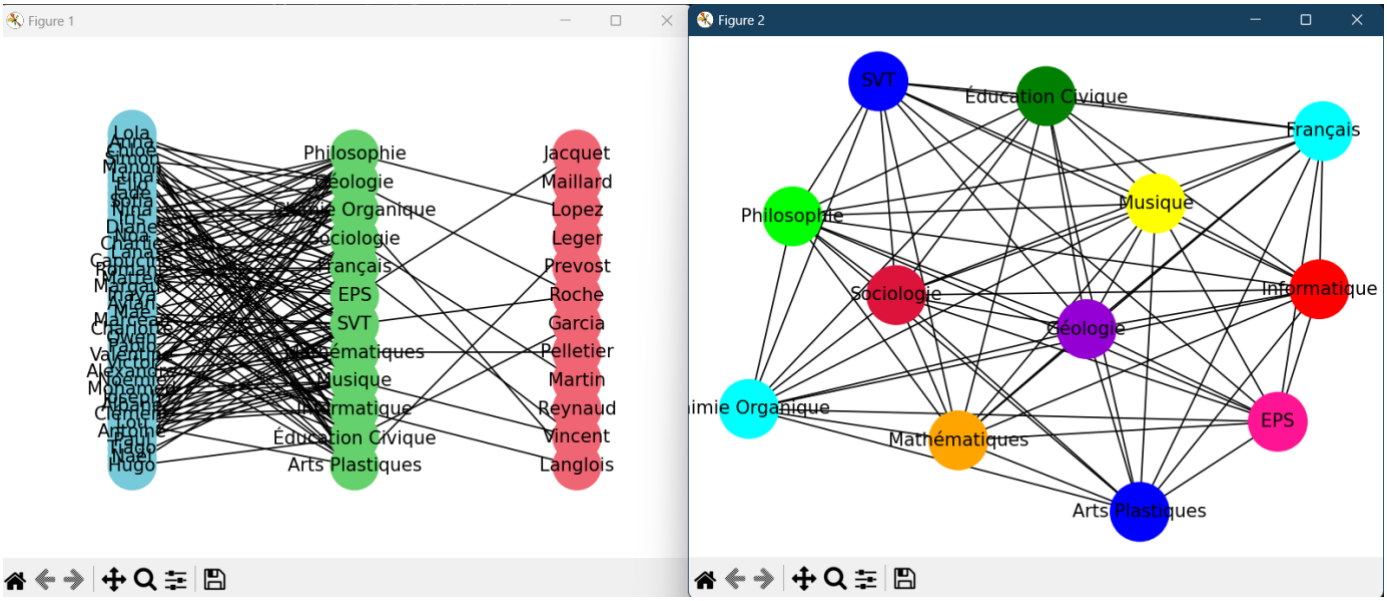

Figure 6 : Graphe représentant la situation finale (gauche) et graphe d'incompatibilité coloré correspondant (droite)

Et voici l'emploi du temps sous forme de tableau correspondant :

Emploi du temps avec 40 élèves, 12 professeurs, 12 matières et 10 créneaux disponibles.

Probabilité de 0.25 entre les élèves et les cours, et de 0.9 sur la disponibilité des professeurs

	Lundi	Mardi	Mercredi	Jeudi	Vendredi
Matin	Arts Plastiques SVT	Informatique	Mathématiques	Français Chimie Organique	Géologie
Midi	Repas	Repas	Repas	Repas	Repas
Après-midi	Éducation Civique	Musique	EPS	Sociologie	Philosophie

Figure 7 : Emploi du temps correspondant à la coloration finale

IV. Conclusion

Pour conclure, on à chercher à appliquer des méthodes de coloration de graphes afin d'optimiser le nombre de créneau nécessaire afin d'élaborer un emploi du temps (si possible) qui rentre dans le nombre de créneau dont l'on dispose.

On a également ajouté des contraintes supplémentaires qui permettent de s'approcher des conditions réelles (Disponibilité variables des professeurs et nombre de salles de cours limités).

Avec plus de temps, on aurait pu chercher à ajouter de nouvelles contraintes ou généraliser certaines situations afin de s'approcher encore davantage de la complexité de l'élaboration d'un emploi du temps scolaire. Voici quelques exemples auquels on a pensé lors de l'élaboration du sujet :

- Possibilité de faire varier la durée des cours :
  On pourrait effectivement définir que certains cours doivent durer un demi-créneau seulement. (A générer avec une certaine probabilité)
- Nécessité pour les élèves de suivre plusieurs fois le même cours chaque semaine :
  Certains cours doivent être suivis plusieurs fois pour avoir suffisamment d'heures d'enseignement.
- Possibilité pour deux professeurs différents de donner le même cours :
  Cela aurait une influence sur la matrice de disponibilité des cours, qui serait désormais l'union des disponibilités des professeurs qui donnent ce cours. De plus, l'incompatibilité généré par un même professeur qui donnent plusieurs cours ne serait plus nécessairement exacte.