SAE 3.02 – Développer des applications utilisant des graphes

Allocation des canaux d'un réseau de bornes Wifi

Matthias DUMAS et Pierre FROSTIN

BUT Réseaux et Télécommunications

Introduction – Cahier des charges

Dans notre SAE « Développer des applications utilisant des graphes », nous avons travaillé sur l'allocation des canaux des bornes Wi-Fi. Comme les réseaux sans fil sont omniprésents et essentiels, il est crucial de les optimiser pour éviter les interférences et améliorer leurs performances.

Nous avons modélisé la répartition des canaux Wi-Fi à l'aide des graphes afin de réduire les conflits de fréquences et d'améliorer le débit et l'expérience utilisateur. Nous avons utilisé l'algorithme **Welsh-Powel**, qui attribue un canal (ou une couleur) à chaque borne Wi-Fi, en tenant compte de la portée des signaux et de la disposition des bornes.

Notre démarche se décompose en deux grandes étapes :

1. **Analyser le problème et le représenter sous forme de graphe :** Nous avons examiné comment les interférences se produisent et comment elles affectent les performances du réseau
2. **Utiliser des algorithmes de coloration de graphes pour optimiser l'allocation des canaux :** Nous avons appliqué l'algorithme Welsh-Powel pour assurer une attribution des canaux, minimisant ainsi les interférences

Ce projet montre comment les graphes peuvent aider à résoudre des problèmes complexes dans les réseaux.

Table des matières

Présentation du sujet

Le projet vise à **concevoir et développer un logiciel** permettant de gérer efficacement un réseau de bornes Wi-Fi. L'objectif principal est d'attribuer des canaux aux différentes bornes en fonction de leurs positions et de leur rayon de couverture, de manière à réduire les interférences et à optimiser les performances du réseau.

Les données d'entrée comprennent les positions cartésiennes (coordonnées X, Y) des bornes Wi-Fi ainsi que le rayon de couverture de chaque borne (distance maximale d'émission du signal). Ces informations seront fournies dans un fichier .csv que le logiciel devra lire.

Le logiciel devra générer un fichier listant les canaux attribués à chaque borne, utilisant les bandes Wi-Fi disponibles (2,4 GHz ou 5 GHz). Ce fichier sera destiné aux techniciens pour configurer les bornes. De plus, le logiciel produira un plan graphique montrant les positions des bornes sur une carte avec les bornes colorées selon leurs canaux respectifs.

Pour répondre à ces exigences, le projet s'appuiera sur la théorie des graphes. Chaque borne est représentée comme un sommet dans un graphe. Une arête est créée entre deux sommets si leurs rayons de couverture se chevauchent, créant ainsi un **graphe d'incompatibilité**.

La méthode de coloration de graphes sera utilisée pour attribuer des canaux, chaque couleur représentant un canal. Le logiciel produira une allocation de canaux cohérente grâce à **l'algorithme de coloration Welsh-Powel.**

Welsh-Powel

L’algorithme de Welsh-Powel colore le graphe en traitant d'abord les sommets dans l'ordre décroissant de leur degré. L'objectif est de commencer par les sommets ayant le plus grand nombre de voisins, car ils sont les plus complexes à colorer. Cela permet de minimiser les conflits dès le départ.

Algorithme de Welsh-Powel

Initialisation

On commence par un dictionnaire appelé `color_map`. Ce dictionnaire est vide au début et servira à associer une couleur à chaque sommet du graphe.

On initialise également une variable appelée `current_color` à 0. Cette variable représente la couleur que l’on tente d’attribuer au sommet courant.

Traitement des sommets par ordre décroissant de degré

On parcourt tous les sommets du graphe, triés par ordre décroissant du nombre de voisins (degré).

L'idée est de commencer par les sommets les plus complexes à colorer, c'est-à-dire ceux qui ont le plus de voisins.

Trouver les couleurs déjà utilisées par les voisins du sommet courant

Pour chaque sommet en cours de traitement :

On examine chacun de ses voisins.

Si un voisin a déjà été coloré (il figure dans le dictionnaire `color_map`), on récupère la couleur qui lui a été attribuée.

Ces couleurs sont placées dans un ensemble appelé `neighbor_colors`, qui permet de stocker uniquement des couleurs uniques (sans doublons).

Attribuer une couleur au sommet courant

On commence avec la couleur `current_color = 0`.

Tant que cette couleur est déjà utilisée par un voisin (c'est-à-dire qu'elle figure dans `neighbor_colors`), on passe à la couleur suivante (`current_color += 1`).

Une fois qu'on a trouvé une couleur libre (non utilisée par les voisins), on l'attribue au sommet courant. Cela se fait en ajoutant une entrée dans le dictionnaire `color_map`, avec le sommet comme clé et la couleur comme valeur.

Réinitialiser la couleur courante

Après avoir attribué une couleur au sommet courant, on réinitialise la variable `current_color` à 0.

Cela permet de recommencer la recherche de couleurs disponibles avec la plus petite couleur possible pour le prochain sommet.

Répéter pour tous les sommets

On répète ce processus pour chaque sommet de la liste triée.

À la fin, tous les sommets auront une couleur attribuée dans le dictionnaire `color_map`.

Exemple simple

L'algorithme que nous vous avons présenté est semblable à celui de Welsh-Powel, à une différence près que nous allons illustrer à travers un exemple.

Voici le graphe que nous utiliserons :

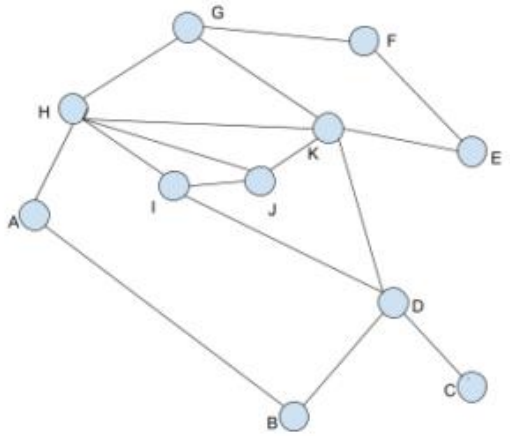

Figure 1 : Graphe non coloré

Tout d'abord, on classe la liste par ordre décroissant de degrés. En cas d'égalité, nous pouvons choisir au hasard n'importe quelle façon de la départager.

Ainsi, le nouvel ordre sera : H, K, D, G, I, J, A, B, E, F, C

Nous examinons les voisins du sommet « H » et vérifions leurs couleurs. Pour l'instant, aucun sommet n'a encore été colorié. Par conséquent, le « set » contenant les couleurs des voisins du sommet « H » est le suivant : `set_H = {}.` Nous attribuons au sommet « H » la couleur 0, qui correspondra au rouge dans cet exemple. Cette association est représentée dans un dictionnaire : `color_map = {H : 0 (R)}.`

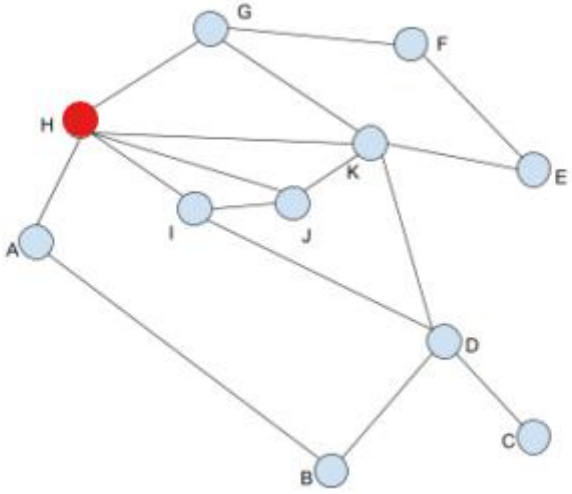

Figure 2 : Coloration du premier nœud (degré le plus élevé)

Ensuite, nous traitons le sommet « K ». Les couleurs de ses voisins sont répertoriées dans `set_K = {0 (R)}.` En parcourant ce « set », nous constatons que la couleur « 0 » (rouge) ne peut pas être attribuée à « K ». Nous passons donc à la couleur suivante « 1 » qui sera, dans cet exemple, le vert.

Mise à jour du dictionnaire : `color_map = {H : 0 (R), K : 1 (V)}.`

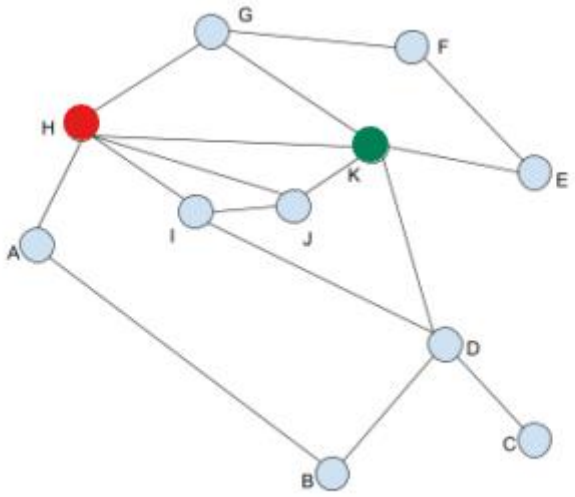

Figure 3 : Coloration du nœud suivant K

Nous appliquons le même processus au sommet « D ». Les couleurs des voisins de « D » sont répertoriées dans `set_D = {1 (V)}.` En parcourant ce « set », nous attribuons à « D » la première couleur disponible, à savoir « 0 », qui correspond au rouge.

Mise à jour du dictionnaire : `color_map = {H:0(R), K:1(V), D:0(R)}.`

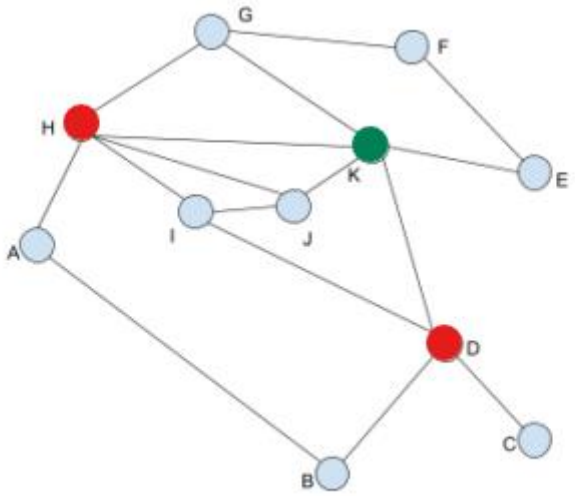

Figure 4 : Coloration du nœud suivant D

Nous passons maintenant au sommet « G » et suivons les mêmes étapes. Les couleurs des voisins sont répertoriées dans `set_G = {0(R), 1(V)}`. Nous attribuons alors au sommet « G » la première couleur disponible. En parcourant ce « set », nous ne constatons qu'aucune des couleurs déjà utilisées n'est appropriée. Nous passons donc à la couleur suivante, la couleur « 2 », qui correspondra ici au bleu.

Mise à jour du dictionnaire : `color_map = {H:0(R), K:1(V), D:1(V), G:2(B)}`.

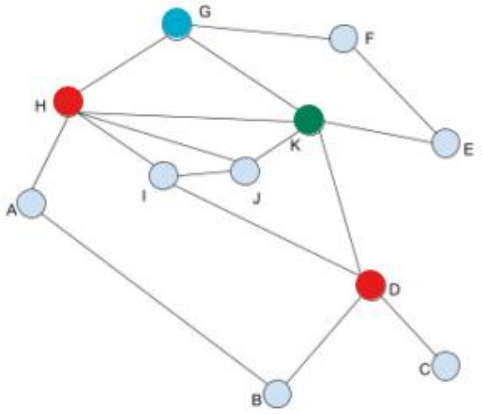

Figure 5 : Coloration du nœuds suivant G

Ainsi de suite nous obtenons ce graphe :

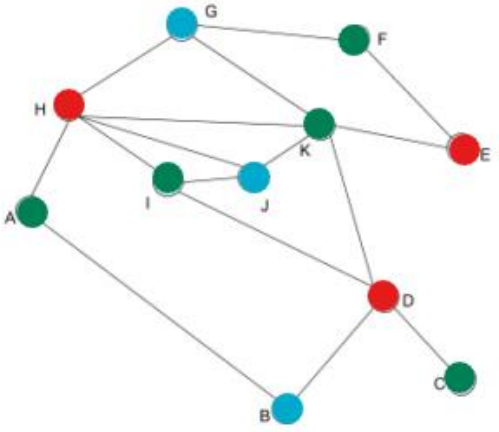

Figure 6 : Graphe entièrement coloré

Nous allons maintenant appliquer l'algorithme « classique » de Welsh-Powel, en utilisant le même graphe que précédemment. Cela permettra de démontrer que, malgré une approche légèrement différente, nous obtenons le même résultat.

Maintenant, en suivant l'algorithme de coloration de Welsh-Powel :

H – colorie en rouge
K – ne colorie pas en rouge, car il est connecté à H
D – colorie en rouge
G – ne colorie pas en rouge, car il est connecté à H
I – ne colorie pas en rouge, car il est connecté à H
J – ne colorie pas en rouge, car il est connecté à H
A – ne colorie pas en rouge, car il est connecté à H
B – ne colorie pas en rouge, car il est connecté à D
E – colorie en rouge
F – ne colorie pas en rouge, car il est connecté à E
C – ne colorie pas en rouge, car il est connecté à D

Après cela, le graphique ressemblera à celui ci-dessous.

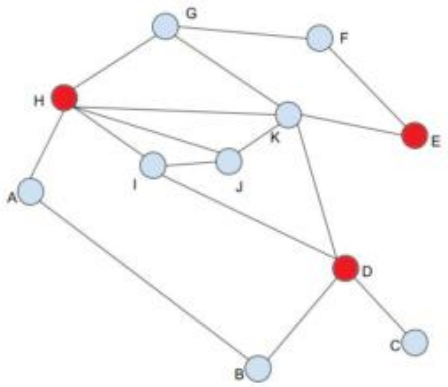

Figure 7 : Coloration des nœuds en rouge

En ignorant les sommets déjà colorés, il nous reste : K, G, I, J, A, B, F, C

Nous pouvons répéter le processus avec la deuxième couleur verte

K – colorie en vert
G – ne colorie pas en vert, car il est lié à K
I – colorie en vert
J – ne colorie pas en vert, car il est lié à I
A – colorie en vert
B – ne colorie pas en vert, car il est lié à A
F – colorie en vert
C – colorie en vert

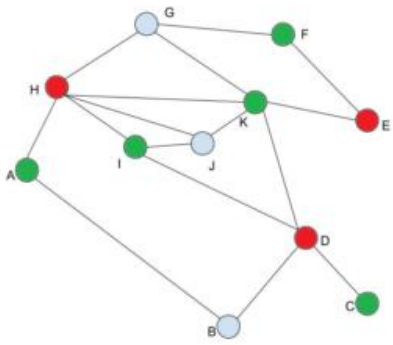

Figure 8 : Coloration des nœuds en vert

Encore une fois, en ignorant les sommets colorés, il nous reste G, J, B. Colorons-les en bleu.

G – couleur bleue

J – couleur bleue

B – couleur bleue

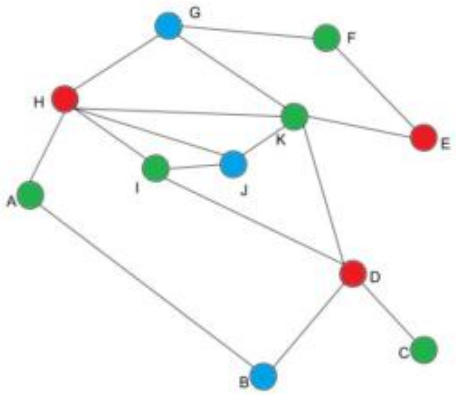

Figure 9 : Coloration des nœuds en bleu – Graphe entièrement coloré

Comme vous pouvez le constater, le résultat obtenu est identique. Cependant, la différence fondamentale réside dans le fait que, dans l'algorithme original de Welsh-Powel, nous parcourons la liste des sommets triés et attribuons la même couleur aux sommets non connectés au sommet déjà coloré, sans examiner directement les voisins du sommet en cours de traitement.

À l'inverse, dans notre algorithme, nous choisissons d'examiner les voisins du sommet en cours. Nous avons opté pour cette approche car elle était plus simple à implémenter au début de notre développement.

Limites de l'algorithme Welsh-Powel

Malgré son efficacité, l'algorithme de coloration Welsh-Powel n'est pas nécessairement le plus optimal. En effet, nous avons trouvé un contre-exemple :

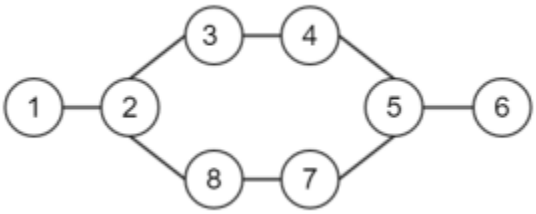

Figure 10 : Contre-exemple

Sommet	Degré	Voisins
1	1	2
2	3	{1,3,8}
3	2	{2,4}
4	2	{3,5}
5	3	{4,6,7}
6	1	5
7	2	{5,8}
8	2	{2,7}

Après application de Welsh-Powel, nous obtenons le graphe coloré suivant :

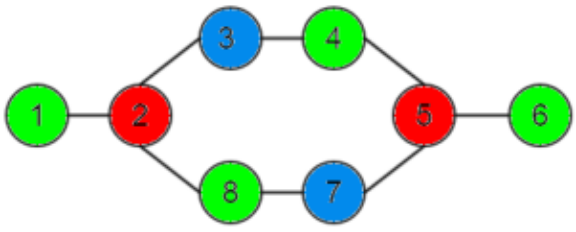

Figure 11 : Contre-exemple coloré avec 3 couleurs d'après Welsh-Powel

Ce graphe utilise donc 3 couleurs (rouge, vert, bleu) pour que chaque nœud ait une couleur propre par rapport à ses voisins. Seulement, voici ce que l'on peut obtenir sans cet algorithme :

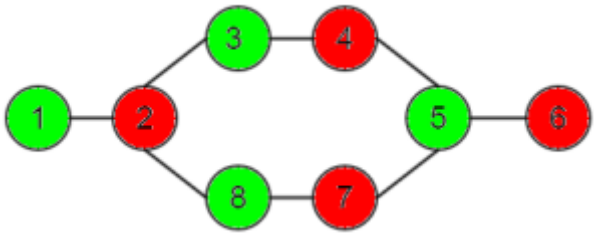

Figure 12 : Contre-exemple coloré avec 2 couleurs

Nous observons donc que, cette fois-ci, nous n'utilisons que deux couleurs. Et pourtant, comme pour Welsh-Powel, aucune couleur n'est identique entre un nœud et son voisin.

Nous avons donc démontré que cet algorithme n'est pas optimal dans certaines situations précises. Dans le cas d'un graphe simple comme celui-ci, identifier le problème n'est pas difficile. Cependant, pour un graphe plus complexe, l'utilisation d'un algorithme de coloration, même imparfait, est toujours plus efficace.

Ce constat nous a donné envie de rechercher un algorithme de coloration plus efficace. Nos recherches se sont soldées par la découverte de l'algorithme de coloration DSATUR *(Degree Saturation)*, créé par Daniel Brélaz en 1979. A l'instar de Welsh-Powel, il s'agit d'un algorithme glouton. Ce dernier n'est pas parfait non plus, mais est plus efficace que Welsh-Powel, avec un taux de réussite de plus de 90%, malgré une vitesse d'exécution moindre.

Si nous avions disposé de plus de temps pour cette SAE, nous aurions pu implémenter cet algorithme. De plus, nous l'avons découvert en fin de projet. Cependant, dans notre cas d'usage, l'algorithme de Welsh-Powell devrait être suffisant.

Néanmoins, nous allons tout de même vous expliquer l'algorithme DSATUR :

Pour chaque sommet, le **degré de saturation** correspond au nombre de couleurs différentes utilisées par ses voisins déjà colorés, un sommet avec un degré de saturation élevé est plus contraint dans son choix de couleurs, car de nombreuses couleurs sont déjà prises par ses voisins. À chaque itération, l'algorithme choisit le sommet avec le degré de saturation maximal pour l'attribuer à une couleur. En cas d'égalité de degré de saturation, on choisit le sommet avec le plus grand degré de voisinage.

Initialisation :

Aucun sommet n’est coloré au début.

Tous les sommets ont un degré de saturation initial de 0.

Choix du premier sommet :

On commence par colorer le sommet avec le degré de voisinage le plus élevé comme pour Welsh-Powel.

Coloration des sommets restants :

À chaque étape, on sélectionne le sommet avec le plus haut degré de saturation.

Si plusieurs sommets ont le même degré de saturation, on choisit celui avec le degré de voisinage le plus élevé.

On attribue au sommet sélectionné la plus petite couleur non utilisée par ses voisins.

Après avoir coloré un sommet, on met à jour les degrés de saturation de ses voisins non colorés.

Le degré de saturation d’un voisin augmente lorsqu’une nouvelle couleur apparaît parmi ses voisins colorés.

Exemple d’utilisation de DSATUR

Pour cela, nous utiliserons le graphe précédent, sur lequel l'algorithme de Welsh-Powell n'est pas optimal.

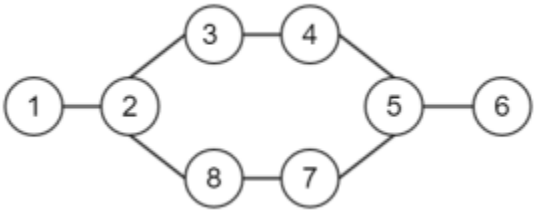

Figure 13 : Graphe non coloré avec DSATUR

Nous commençons par classer les sommets en fonction de leur degré. Comme les sommets « 5 » et « 2 » ont le même degré, nous commençons arbitrairement par le sommet « 5 ».

Nous colorons le sommet « 5 » en vert :

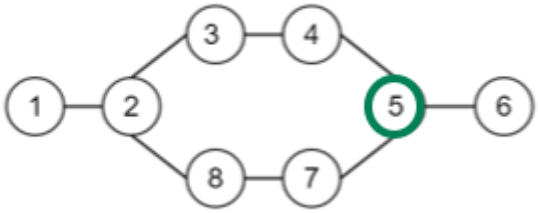

Figure 14 : Coloration du nœuds 5 (plus grand degré) en vert

Nous examinons ensuite le sommet ayant le **degré de saturation** (DSAT) le plus élevé. Ici, les sommets « 4 », « 7 » et « 6 » ont le même DSAT. Comme ces trois sommets ont le même DSAT, nous devons choisir celui ayant le degré le plus élevé. Or, nous remarquons que « 4 » et « 7 » ont le même degré, nous choisissons donc arbitrairement le sommet « 4 », que nous colorons en rouge.

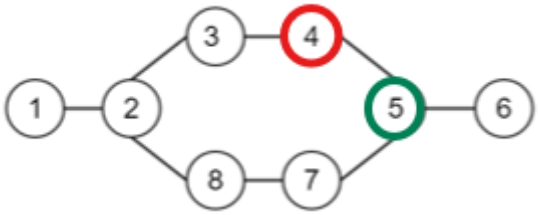

Figure 15 : Coloration du voisin de plus grand degré 4 en rouge (vert impossible)

Nous répétons les étapes, « 3 », « 7 » et « 6 » ont le même DSAT, nous prenons le sommet avec le degré le plus élevé, « 3 » et « 7 » ayant le même degré nous choisissons arbitrairement le sommet « 3 » que nous colorons avec la première couleur disponible le vert.

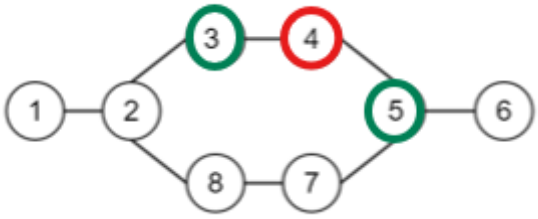

Figure 16 : Coloration du voisin de 4 en vert

Les sommets « 2 », « 7 » et « 6 » ont le même **DSAT**. Cependant, « 2 » a le plus grand nombre de voisins, nous le colorons donc avec la première couleur disponible, le rouge.

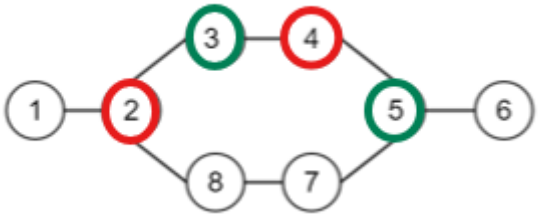

Figure 17 : Coloration du voisin de 3 en rouge (vert impossible)

Les sommets « 1 », « 8 », « 7 » et « 6 » ont le même **DSAT**. Cependant, « 8 » et « 7 » ont le plus grand nombre de voisins, nous choisissons arbitrairement le sommet « 8 » nous le colorons donc avec la première couleur disponible, le vert.

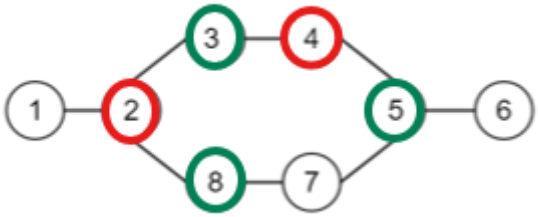

Figure 18 : Coloration du voisin de 2 avec le plus grand degré en vert

Ensuite le sommet « 7 » a le plus grand DSAT nous le colorons donc avec la première couleur disponible : le rouge

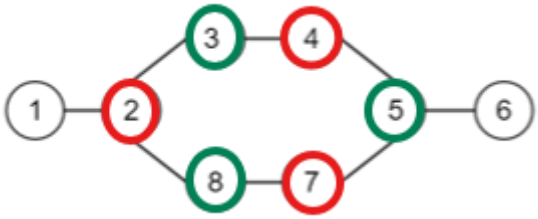

Figure 19 : Coloration du voisin de 8 en rouge (vert impossible)

Enfin nous effectuons les mêmes étapes que précédemment et voici ce que nous obtenons :

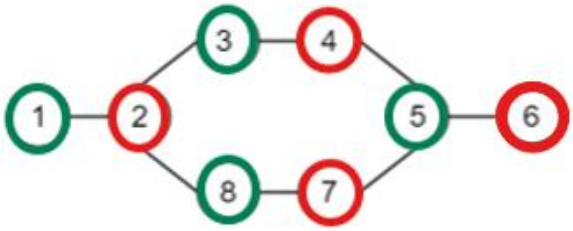

Figure 20 : Coloration de 1 et 6 avec la couleur non utilisée par leur voisin

Ainsi nous remarquons que l'algorithme DSATUR est plus optimale que Welsh-Powel car celui-ci peut minimiser le nombre de couleurs utilisées.

Limites du logiciel et extensions

Avec environ 25h de projet dédiées au code, nous avons dû nous limiter à certaines fonctionnalités. Voici les fonctionnalités implémentées :

- Génération d'un graphe avec les canaux 5GHz si le nombre de canaux nécessaires en 2,4GHz est supérieur à 3
- Coloration des rayons d'émission des différents points d'accès
- Génération forcée du 2,4GHz demandée à l'utilisateur même si le nombre de canaux nécessaires est trop élevé
- Affichage de la légende des canaux utilisés pour chaque AP
- Génération de fichiers .csv contenant la liste des APs ainsi que leur canal défini par le logiciel

Nous pourrions ajouter les fonctionnalités suivantes :

- Afficher un plan « fond de carte » sur l'affichage des APs
- Autoriser un degré d'erreur défini par l'utilisateur concernant le chevauchement des surfaces d'émission (autoriser par exemple 2m de chevauchement de rayon entre deux APs pour réduire le nombre de canaux nécessaires au déploiement)
- Créer une interface graphique plus conviviale

Gestion de projet

Voici notre gestion de projet de base :

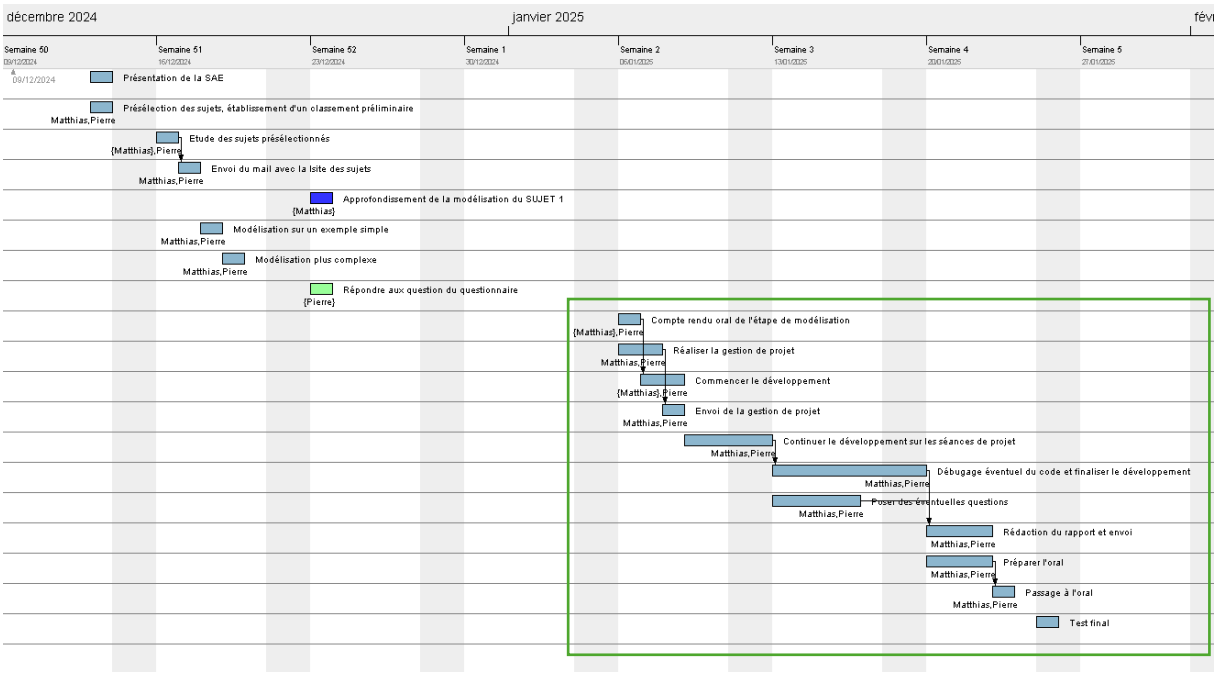

Figure 21 : Gestion de projet de base

Pour notre gestion de projet finale, nous nous concentrerons uniquement sur la partie encadrée en vert.

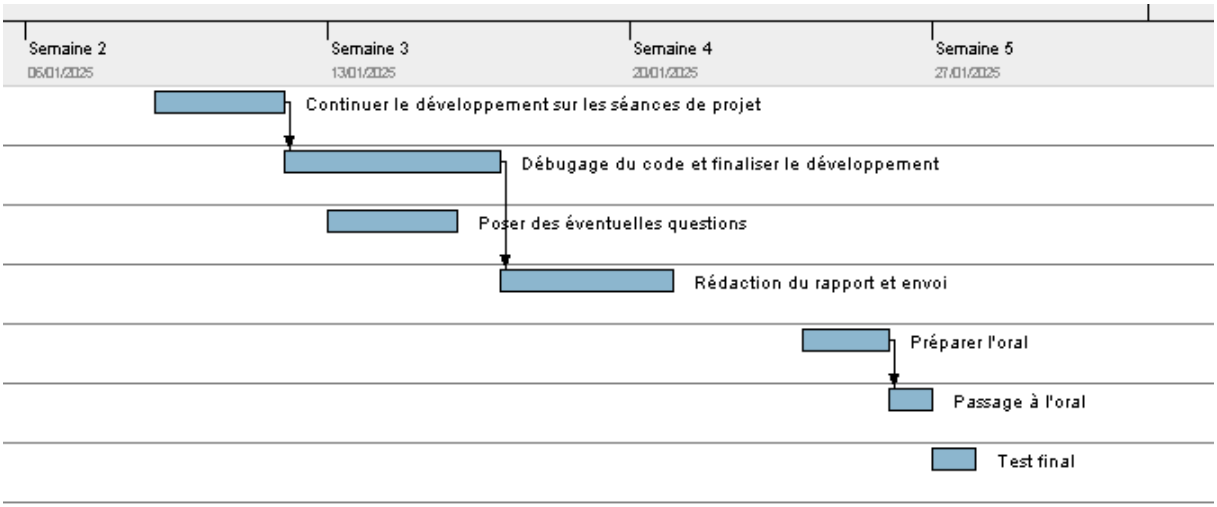

Figure 22 : Gestion de projet à la fin de la réalisation

Tâche : Continuer le développement sur les séances de projet

- Finaliser l'algo
- Lire le fichier avec les données
- Récupérer les coordonnées cartésiennes
- Afficher le graphe d'incompatibilité
- Effectuer des tests pour un graphe simple
- Effectuer des tests pour un graphe plus complexe
- Compléter l'algo de Welsh-Powell
- L'algo a pris plus de temps que prévu

Tâche : Débogage du code et finaliser le développement

- Programmer Welsh-Powell (lors de la programmation nous avons légèrement modifier l'algo pour enlever certaine parti inutile)
- Test du code sur des graphes simples puis plus complexe
- Code une fonction qui propose à l'utilisateur en cas de chevauchement des canaux de passer de 2.4 GHz à 5GHz
- En cas de nombre de canaux supérieur à 7 forcer l'utilisateur à utiliser la bande 5GHz
- Réification (cela a été plus long que prévu car nous ne devions nous adapter au code de l'autre)

Tâche : Poser des éventuelles questions

- Demander des précisions sur le contenu du rapport et plus spécifiquement sur la partie cœur de projet
- Demander des précisions sur l'oral à passer

Conclusion

Pour conclure, cette SAE nous a permis de mettre en pratique notre compréhension théorique des graphes dans un projet concret, nous fournissant des moyens efficaces pour résoudre une problématique liée aux réseaux. Nous avons développé nos compétences en utilisant la bibliothèque NetworkX de Python et avons également implémenté l'algorithme de coloration de Welsh-Powel.

Ce projet vise à développer un logiciel innovant pour la gestion optimale des réseaux de bornes Wi-Fi en utilisant la théorie des graphes et l'algorithme de coloration de Welsh-Powel. Grâce à ce logiciel, les techniciens pourront attribuer des canaux de manière efficace, réduire les interférences et maximiser les performances du réseau. Le résultat final inclura un fichier de configuration des canaux ainsi qu'une carte graphique des bornes, offrant une solution pratique et visuelle pour la gestion des réseaux Wi-Fi.

Table des illustrations

Sources

Fonctionnement de Welsh-Powel et contre-exemple

Coloration de graphe - Wikipédia

Documentation NetworkX

Documentation Matplotlib

Positionnement de la légende

Modification de la taille de l'affichage

Algorithme DSATUR

Welsh-Powel et DSATUR