Projet de Coloration de graphes

Réalisé par :

- STIEVENARD Emma
- MEZOUAR Chahinez

Table des matières :

1. Introduction

1.1 Définition du problème

1.1.1 Qu'est-ce qu'un graphe ?

Un graphe G est défini par le couple (V,E) où V désigne l'ensemble des sommets et $E \subseteq V \times V$ l'ensemble des arêtes. La coloration de graphe constitue une fonction $f:V \rightarrow C$ où C représente l'ensemble des couleurs, telle que $\forall (u,v) \in E$, $f(u) \neq f(v)$.

1.1.2 Coloration de graphe : définition et contraintes

La coloration de graphe consiste à affecter une couleur distincte à chaque sommet du graphe en respectant plusieurs contraintes fondamentales. Tout d'abord, chaque sommet doit recevoir exactement une couleur, ce qui garantit l'unicité. Ensuite, deux sommets adjacents, c'est-à-dire reliés par une arête, doivent porter des couleurs différentes, ce qui constitue la contrainte de non-adjacence. Enfin, l'objectif est de minimiser le nombre total de couleurs utilisées, appelé le nombre chromatique X(G).

1.2 Applications de la coloration de graphe

La coloration de graphe trouve des applications dans divers domaines. Dans l'optimisation de compilation, elle est utilisée pour l'allocation efficace des registres dans les compilateurs. Dans la gestion des ressources, elle permet de minimiser les conflits lors du partage des ressources, par exemple dans la planification de tâches ou le transport. Enfin, elle est également utilisée dans des cas concrets comme la gestion des horaires ou la résolution de problèmes dans les réseaux de télécommunications.

1.3 Objectifs du projet

Les principaux objectifs de ce projet sont, d'une part, de réduire le nombre de couleurs nécessaires pour colorer un graphe donné tout en respectant les

contraintes de coloration. D'autre part, il s'agit d'évaluer les performances des algorithmes appliqués à des graphes variés pour identifier les solutions les plus efficaces.

2. Cadre Théorique

- ## 2.1 Notions fondamentales
  - ### 2.1.1 Sommets, arcs et graphes orientés/non orientés

    Les graphes peuvent être classés en deux grandes catégories. Les graphes orientés possèdent des arêtes avec une direction définie reliant un sommet source à un sommet destination. À l'inverse, les graphes non orientés ont des arêtes symétriques sans direction spécifique.

  - ### 2.1.2 Propriétés des graphes

    Parmi les propriétés importantes des graphes, on trouve la connexité, qui indique si tous les sommets sont reliés directement ou indirectement, et les cycles, qui sont des chemins fermés dans le graphe. Le nombre chromatique est une propriété clé liée à la coloration.

- ## 2.2 Méthodes de résolution
  - ### 2.2.1 Définition des algorithmes de base

    Les **algorithmes de base** sont ceux qui suivent une méthode systématique pour trouver une solution, souvent avec des garanties d'optimalité, mais peuvent être coûteux en termes de temps de calcul pour de grands graphes. Parmi ces algorithmes, on trouve des techniques classiques telles que :

    - **L'algorithme de coloration gloutonne:** Il attribue des couleurs aux sommets de manière séquentielle en cherchant à minimiser les conflits à chaque étape.
    - **L'algorithme de Welsh-Powell :** Il trie les sommets par ordre décroissant de degré et applique une coloration gloutonne en respectant cet ordre.

  - ### 2.2.2 Approches heuristiques et exactes

    Les **approches heuristiques** visent à trouver une solution bonne (mais non nécessairement optimale) de manière rapide. Elles sont souvent utilisées pour résoudre des problèmes NP-difficiles, comme la coloration de graphe, dans des délais raisonnables.

    Exemple :

- **Hill-Climbing** qui explore des solutions voisines pour essayer d'améliorer l'état actuel sans garantie d'optimalité.
- **Algorithmes génétiques**, qui utilisent des techniques de sélection, croisement et mutation pour explorer efficacement l'espace des solutions.

3. Travail Préliminaire

3.1 Recherche documentaire

3.1.1 Identification des sources

La recherche a débuté par l'identification des sources pertinentes pour le sujet étudié. Cela inclut :

- **Thèses académiques** :

  Cyril GRELIER. (2023). *Métaheuristiques Guidées par l'Apprentissage pour la Coloration de Graphe*(Thèse de doctorat). Université d'Angers. "https://theses.hal.science/tel-04523127/document"

  Romain Montagné . (2016). *Optimisation de l'allocation de ressources dans un réseau de télécommunications par coloration impropre de graphes*(Mémoire ou thèse).Polytechnique Montréal. "https://publications.polymtl.ca/2112/1/2016_RomainMontagne.pdf"

- **Diapositives de cours :**

  (2017) *INF4230 – Intelligence Artificielle. Recherche locale*. Université du Québec à Montréal. "https://gdac.uqam.ca/inf4230/diapos/05-recherche-locale.pdf"

  Laurent Moalic. (2018). *Coloration de graphe méthodes et applications.* Académie de Strasbourg, France. "https://pedagogie.ac-strasbourg.fr/fileadmin/pedagogie/isn/ISN/Conferences/cours_isn_Coloration_graphe.pdf"

3.2 Analyse des besoins

3.2.1 Structure de données pour les graphes

La classe `GrapheReel` utilise les structures de données suivantes :

1. Représentation des nœuds :
   - Un ensemble (`set`) nommé `self.noeuds` qui stocke les identifiants des nœuds
   - Les identifiants peuvent être des chaînes de caractères ou des nombres

        - L'utilisation d'un `set` garantit l'unicité des nœuds et permet un accès rapide en O(1)
    2. Représentation des arêtes :
        - Une liste (`list`) nommée `self.aretes` qui stocke les arêtes sous forme de tuples (source, destination)
        - Pour un graphe non orienté, chaque arête est stockée deux fois dans les deux directions
        - Pour un graphe orienté, chaque arête n'est stockée qu'une fois dans sa direction
    3. Attributs supplémentaires :
        - `self.oriente` : booléen qui indique si le graphe est orienté ou non
        - `self.coloration_actuelle` : dictionnaire qui associe à chaque nœud sa couleur (pour la coloration de graphe)
- 3.2.2 Formats d'entrée et de sortie
    1. Format DIMACS pour les fichiers :
        - Les lignes commençant par 'c' sont des commentaires
        - La ligne 'p' définit le nombre de nœuds et d'arêtes
        - Chaque ligne 'e' définit une arête entre deux nœuds
    2. Format de sortie pour la coloration :
        - Un nœud et sa couleur par ligne
        - Les couleurs sont représentées en format hexadécimal (#RRGGBB)
    3. Sortie visuelle :
        - Génération d'images PNG via Graphviz
        - Les nœuds sont représentés par des cercles
        - Les arêtes sont représentées par des lignes/flèches
        - Les couleurs de la coloration sont appliquées aux nœuds
        - Utilisation du moteur 'neato' pour le placement des nœuds
    4. Format des statistiques de coloration :
        - Retourne les performances et résultats des algorithmes de coloration

3.3 Méthodologie de travail

3.3.1 Répartition des tâches

Le projet a été organisé en différentes étapes, comprenant la documentation, l'implémentation, les tests et la rédaction du rapport de projet.

Tout d'abord, chacune de notre côté nous nous sommes documentées sur le problème à traiter et les différents algorithmes proposés pour le résoudre. Ensuite, nous nous sommes concertées sur l'algorithme que nous avions choisi de réaliser et nous l'avons fait. Nous nous sommes aidées si l'une de nous deux était bloquée. Puis, une fois les algorithmes réalisés, nous avons fait un programme général permettant de les tester ainsi que de répondre aux autres besoins (créer, charger, afficher un graphe et évaluer une coloration). Enfin, nous avons rédigé ensemble le rapport du projet.

4. Développement des Solutions

- 4.1 Structure de données pour les graphes
  - 4.1.1 Lecture et écriture de graphes
    1. Lecture de graphes (méthode `lire_fichier`):
       - Lit un fichier au format DIMACS
       - Traitement ligne par ligne avec gestion des différents types :
         - 'c' : lignes de commentaires ignorées
         - 'p' : définition du problème (nombre de nœuds et arêtes)
         - 'e' : définition des arêtes
       - Vérification de la cohérence des données lues avec les valeurs déclarées
       - Gestion des erreurs pour les formats invalides
    2. Écriture de graphes (méthode `ecrire_fichier`):
       - Sauvegarde au format DIMACS
       - Génère un en-tête avec commentaire
       - Écrit la ligne de problème avec le nombre de nœuds et d'arêtes
       - Gestion des erreurs d'écriture
       - Pour les graphes non orientés :
         - Évite les doublons d'arêtes
         - Utilise `tuple(sorted([source, dest]))` pour normaliser l'ordre
  - 4.1.2 Modification et manipulation
    1. Gestion des nœuds :
       - `ajouter_noeud(identifiant)` : Ajoute un nœud à l'ensemble self.noeuds
       - `supprimer_noeud(identifiant)` :
         - Retire le nœud de l'ensemble
         - Supprime toutes les arêtes associées
         - Met à jour la coloration si elle existe
    2. Gestion des arêtes :
       - `ajouter_arete(source, destination)` :
         - Vérifie l'existence des nœuds
         - Pour les graphes non orientés, ajoute l'arête dans les deux sens
       - `supprimer_arete(source, destination)` :
         - Supprime l'arête spécifiée
         - Pour les graphes non orientés, supprime aussi l'arête inverse
    3. Fonctions auxiliaires :
       - `get_voisins(noeud)` : Retourne l'ensemble des voisins d'un nœud

        - Pour les graphes orientés : union des voisins entrants et sortants
        - Pour les graphes non orientés : tous les nœuds connectés
    - ❖ `creer_graphe_interactif()` : Interface utilisateur pour créer un graphe manuellement
        - Choix du type de graphe (orienté/non orienté)
        - Saisie interactive des nœuds et des arêtes

- 4.2 Algorithmes développés
    - 4.2.1 Welsh-Powell

L'algorithme de Welsh-Powell est une méthode gloutonne utilisée pour résoudre le problème de la coloration de graphes. Il vise à minimiser le nombre de couleurs utilisées pour colorer les sommets tout en respectant la contrainte de non-adjacence (deux sommets reliés ne peuvent pas avoir la même couleur).

Étapes

1. **Trier les sommets :** Les sommets du graphe sont triés par degré décroissant (nombre de voisins). Les sommets ayant le plus de connexions sont traités en premier.
2. **Colorer les sommets :**
    - On initialise une liste de couleurs disponibles.
    - On parcours chaque sommet dans l'ordre trié :
        - On choisit la couleur la plus petite possible qui n'est pas déjà utilisée par ses voisins.
        - On assigne cette couleur au sommet.
3. **On répète cela jusqu'à ce que tous les sommets soient colorés.**

Avantages

- C’est simple à implémenter.
- C’est performant pour des graphes peu denses ou avec des degrés bien répartis.

Inconvénients

- La solution trouvée n'est pas toujours optimale, surtout pour des graphes très irréguliers.
- Cela peut utiliser plus de couleurs que le nombre chromatique minimal.

    - 4.2.2 Hill-climbing

Principe

Le Hill-Climbing est une approche heuristique utilisée pour trouver une solution en explorant les voisins d'une solution actuelle et en se déplaçant vers une solution meilleure. Dans le cadre de la coloration de graphes, il tente de minimiser le nombre de conflits entre les sommets tout en réduisant le nombre total de couleurs.

Étapes

1. **Initialisation :**
    - On génère une solution initiale, par exemple en assignant des couleurs aléatoires aux sommets.
2. **Recherche locale :**
    - On calcule une fonction d'évaluation (e.g., le nombre de conflits ou le nombre de couleurs utilisées).
    - On modifie la coloration (changer la couleur d'un sommet ou échanger des couleurs).
    - Si une amélioration est trouvée, on accepte le changement.
3. **Répéter :**
    - On continue à modifier la solution jusqu'à ce qu'aucune amélioration ne soit possible ou qu'un critère d'arrêt soit atteint (par exemple, un temps maximum).
4. **Renvoi de la solution :**
    - La meilleure solution rencontrée est retournée.

Avantages

- Flexible : Il peut être utilisé pour différents types de graphes.
- C'est une approche rapide pour obtenir une solution acceptable.
- Il est facile à combiner avec d'autres méthodes comme Tabu Search.

Inconvénients

- Il peut converger vers un optimum local, qui n'est pas la solution optimale.
- Il nécessite une bonne définition de la fonction d'évaluation et de voisinage.

4.3 Comparaison des algorithmes

4.3.1 Critères de performance

- **Efficacité** : Welsh-Powell est plus rapide pour résoudre un problème de coloration, tandis que Hill-Climbing peut être plus lent en raison des multiples itérations nécessaires.
- **Qualité des solutions** : Hill-Climbing peut trouver de meilleures solutions, mais risque de rester bloqué dans des optima locaux. Welsh-Powell offre des solutions rapides mais parfois sous-optimales.

- **Robustesse** : Welsh-Powell est plus constant dans ses résultats, contrairement à Hill-Climbing, qui dépend fortement du point de départ.

5. Implémentation Technique

5.1 Guide d'utilisation du projet

Pour utiliser notre code et faire fonctionner le projet, voici les instructions à suivre/

Pour commencer, il faut dézipper notre dossier contenant les fichiers de notre projet. Puis, il faut ouvrir ce dossier dans un éditeur de code. Vous ouvrez le terminal (console) et vous y écrivez : py graphe6.py

Le fichier se lance donc et un menu apparaît. Voici les différentes fonctionnalités du menu et comment il fonctionne :

1. Créer un nouveau graphe : Cette fonctionnalité sert à créer un graphe personnalisé en répondant à plusieurs questions. Tout d'abord, si le graphe est orienté ou non. Ensuite on saisit le nom d'autant de nœuds qu'on le souhaite. Une fois fini, on clique sur la touche "Entrez" du clavier pour passer à l'étape suivante. On saisit les arêtes de la même manière. Puis le graphe est créé

2. Charger un graphe depuis un fichier : Cette deuxième fonctionnalité permet de charger un graphe contenu dans le dossier de notre projet. Par exemple, saisissez "myciel5.col" pour charger ce graphe.

3. Sauvegardez le graphe : Ici, on peut sauvegarder le graphe créer avec la fonctionnalité du menu. Il suffit de rentrer son nom et il sera sauvegardé dans le dossier actuel avec le nom souhaité au format DIMACS.

4. Ajouter un nœud : Sur le graphe en cours de création, si vous souhaitez y ajouter un nouveau nœud, tapez simplement le nom de ce nouveau nœud et il s'ajoutera au graphe que vous êtes en train de créer.

5. Supprimer un nœud : De la même manière, vous pouvez supprimer un nœud déjà créé sur votre graphe en tapant simplement son identifiant (nom que vous lui avez donné).

6. Ajouter une arête : Toujours sur le graphe en cours, vous pouvez ajouter une arête entre deux nœuds existants. Par exemple entre deux nœuds créés avec la fonctionnalité 4. Vous entrez le nœud source de l'arête et destination pour faire cela.

7. Supprimer une arête : Enfin, vous pouvez supprimer une arête du graphe existant de la même manière qu'on supprime un nœud à la fonctionnalité.

8. Afficher le graphe : Cette fonctionnalité affiche le graphe en cours dans la console de la manière suivante : Noeuds : [ ] Arêtes : [ ]. Avec les valeurs contenues entre guillemets dans les crochets et séparées par des virgules.

9. Créer une coloration avec l'algorithme Welsh Powell : Cette fonctionnalité colore le graphe chargé (ou créé) avec l'algorithme Welsh Powell que nous avons développé. Dans la console, cette fonctionnalité vous donnera les détails des couleurs de la coloration créée. Puis, des statistiques telles que le temps d'exécution, le nombre d'itérations et le nombre de conflits finaux.

10. Charger une coloration : Ici vous pouvez charger un fichier de coloration se trouvant dans le dossier courant. Vous avez simplement à entrer le nom du fichier de coloration souhaité.

11. Sauvegarder la coloration : Cette fonctionnalité permet d'enregistrer la coloration créée avec l'algorithme des fonctionnalités 9 ou 14 dans un fichier dans le dossier courant. Cela peut notamment être utile pour charger cette coloration dans la fonctionnalité 10 si besoin.

12. Évaluer une coloration : Ici, le programme analyse la coloration réalisée et indique si elle est valide et combien de couleurs elle utilise.

13. Visualiser le graphe (PNG) : Cette fonctionnalité enregistre le graphe créé avec les étapes précédentes au format PNG dans le dossier courant.

14. Colorer le graphe (Hill-Climbing): Cette fonctionnalité colore le graphe chargé (ou créé) avec l'algorithme Hill Climbing que nous avons développé. Dans la console, cette fonctionnalité vous donnera les mêmes détails des couleurs de la coloration créée qu'avec l'algorithme Welsh-Powell.

0. Quitter : Enfin, avec cette fonctionnalité vous pouvez quitter le menu.

- 5.1 Construction des graphes de test

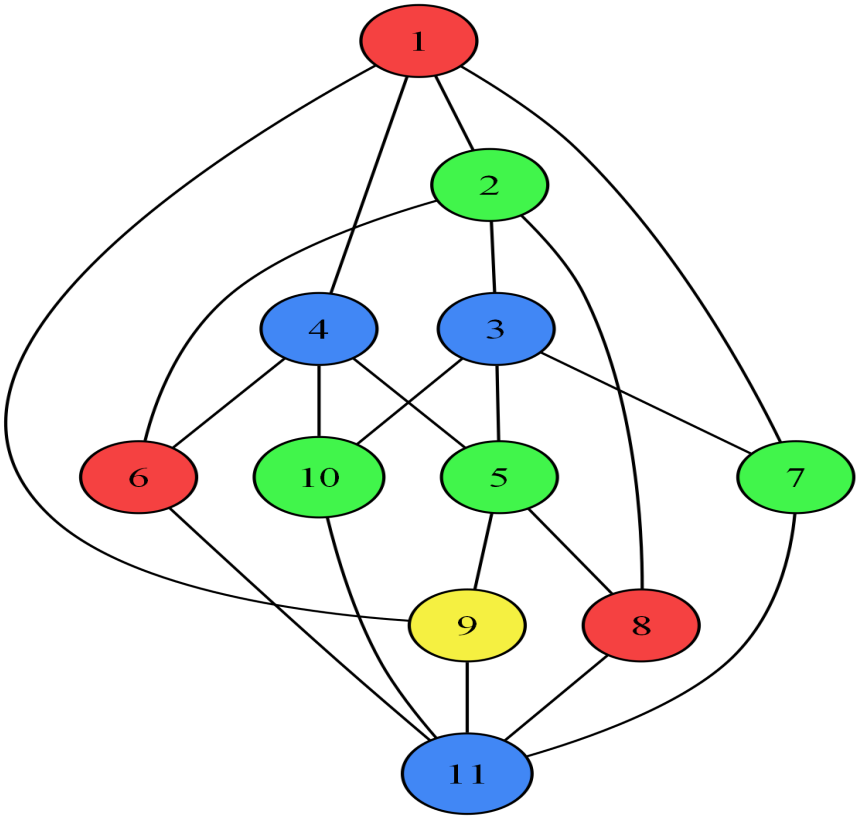
  Des graphes de différentes tailles ont été construits pour tester les algorithmes. Cela inclut des graphes simples, comptant jusqu'à 10 sommets, ainsi que des graphes complexes avec plus de 100 sommets. Il est également possible de charger et construire des graphes, ainsi que de modifier et supprimer des arêtes et des sommets, permettant une flexibilité maximale pour tester différentes configurations et scénarios.

- 5.2 Résultats expérimentaux
  - 5.2.1 Présentation des résultats
  -

1. Première expérience : Fichier myciel3.col

```
Votre choix : 2
Nom du fichier à charger : myciel3.col
Graphe lu: 11 nœuds, 20 arêtes
Graphe chargé : 11 noeuds, 20 arêtes
```

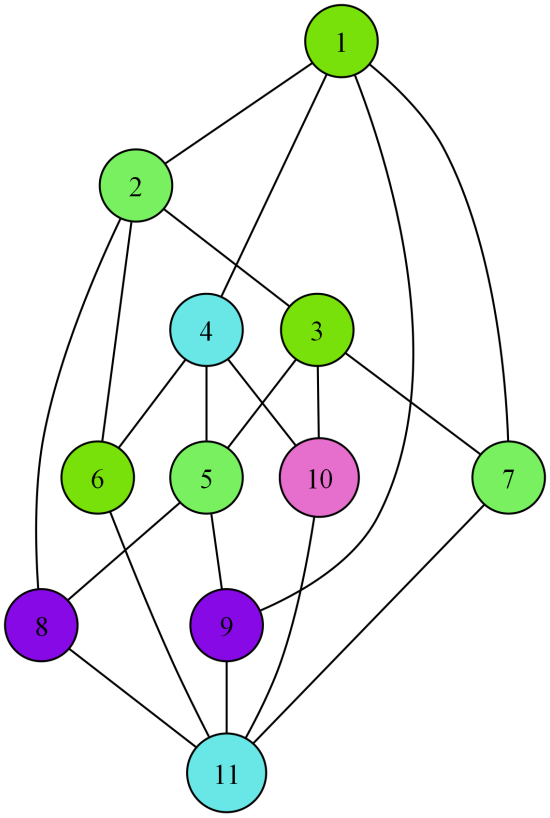
Coloration avec l'algorithme Welsh-Powell :




```
Statistiques:
Temps d'exécution: 0.0 secondes
Nombre d'itérations: 1
Conflits finaux: 0
```

```
Votre choix : 12
La coloration est valide et utilise 4 couleur(s).
```

Coloration avec l'algorithme Hill-Climbing




```
Statistiques:
Temps d'exécution: 0.0015873909 secondes
Nombre d'itérations: 1
Conflits finaux: 0
```

```
Votre choix : 12
La coloration est valide et utilise 5 couleur(s).
```

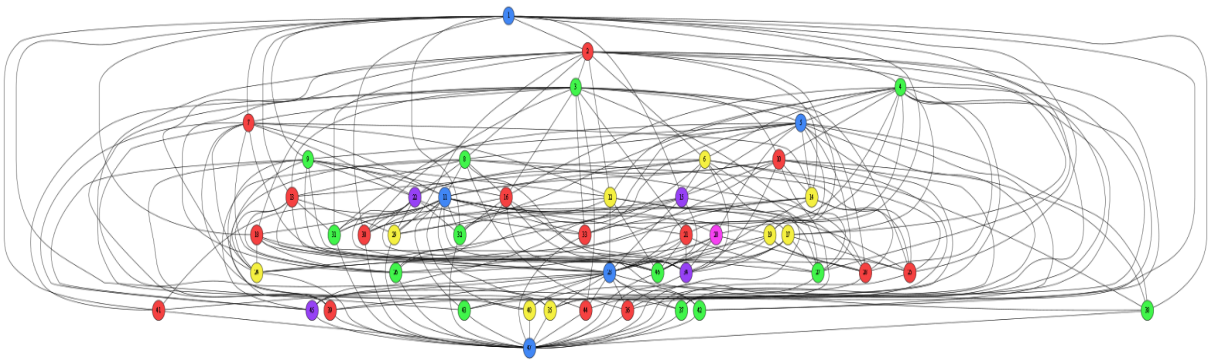
Les deux algorithmes ont réussi à produire des solutions correctes, c'est-à-dire sans conflits entre les couleurs des sommets connectés par une arête. Cela confirme leur efficacité dans le respect des contraintes de coloration. Toutefois, des différences ont été observées dans le nombre total de couleurs utilisées. Welsh-Powell a nécessité 6 couleurs, tandis que Hill-Climbing a pu obtenir une solution légèrement meilleure avec seulement 5 couleurs.

En termes de rapidité, une différence a été constatée. L'algorithme Welsh-Powell a exécuté son traitement presque instantanément, avec un temps mesuré à 0.0 secondes, alors que Hill-Climbing a nécessité un temps légèrement supérieur de 0.001 secondes.

2. Deuxième expérience : Fichier myciel5.col

```
Votre choix : 2
Nom du fichier à charger : myciel5.col
Graphe lu: 47 nœuds, 236 arêtes
Graphe chargé : 47 noeuds, 236 arêtes
```

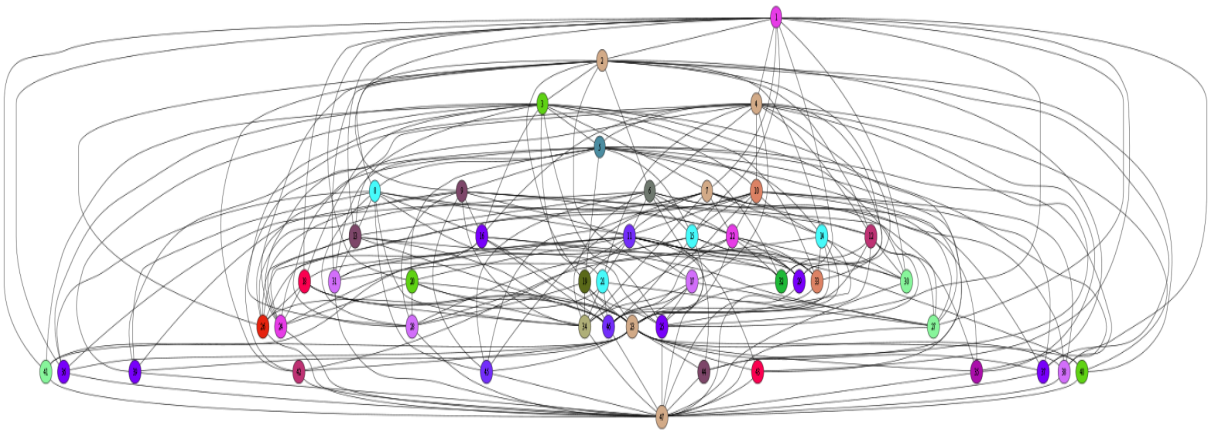
Coloration avec l'algorithme Welsh-Powell :



```
Statistiques:
Temps d'exécution: 0.001619339 secondes
Nombre d'itérations: 1
Conflits finaux: 0
```

```
Votre choix : 12
La coloration est valide et utilise 6 couleur(s).
```

Coloration avec l'algorithme Hill-Climbing



```
Statistiques:
Temps d'exécution: 0.0675902367 secondes
Nombre d'itérations: 1
Conflits finaux: 0
```

```
Votre choix : 12
La coloration est valide et utilise 19 couleur(s).
```

Lors du deuxième test, une différence notable est apparue entre les deux algorithmes, tant au niveau du temps d'exécution que du nombre de couleurs utilisées pour résoudre le problème de coloration de graphes.

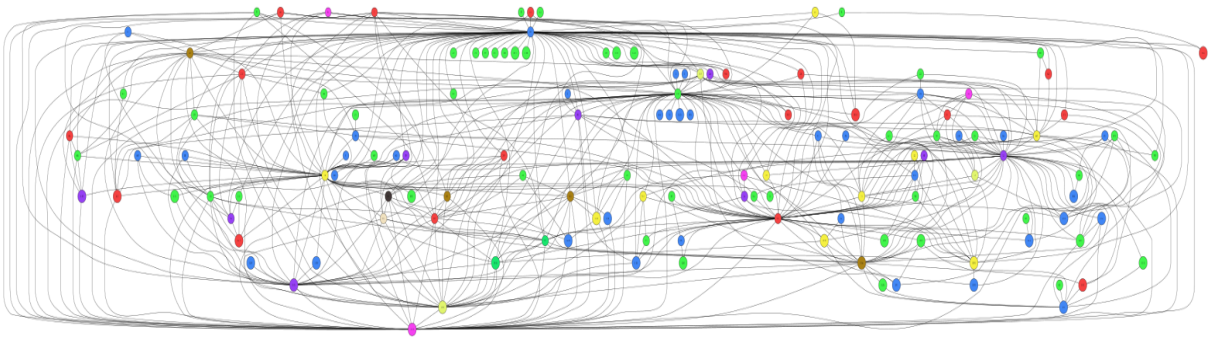
Welsh-Powell s'est avéré particulièrement efficace, avec un temps d'exécution d'un peu plus de 0,01 seconde. En revanche, Hill-Climbing a été significativement plus lent, nécessitant plus de 0,06 seconde pour parvenir à une solution.

Welsh-Powell a réussi à produire une solution avec seulement 6 couleurs, démontrant ainsi son efficacité dans la minimisation des ressources pour ce test. Hill-Climbing, en revanche, a utilisé 19 couleurs, un résultat bien moins optimisé et significativement plus coûteux en termes de ressources.

3. Troisième expérience : Fichier anna.col

```
Votre choix : 2
Nom du fichier à charger : anna.col
Graphe lu: 138 nœuds, 986 arêtes
Graphe chargé : 138 noeuds, 986 arêtes
```

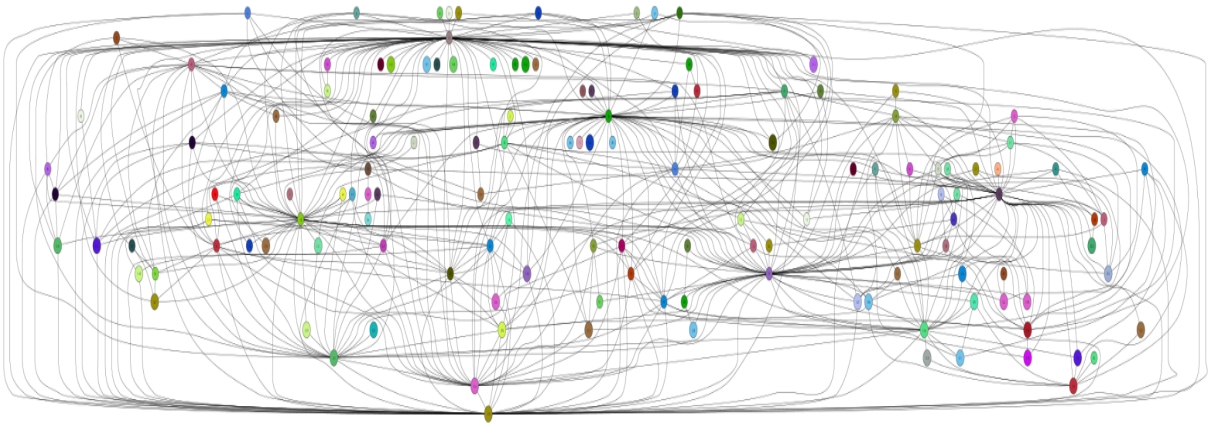
Coloration avec l'algorithme Welsh-Powell :



```
Statistiques:
Temps d'exécution: 0.0243489742 secondes
Nombre d'itérations: 1
Conflits finaux: 0
```

```
Votre choix : 12
La coloration est valide et utilise 11 couleur(s).
```

Coloration avec l'algorithme Hill-Climbing



```
Statistiques:
Temps d'exécution: 1.3813998699 secondes
Nombre d'itérations: 1
Conflits finaux: 0
```

```
Votre choix : 12
La coloration est valide et utilise 60 couleur(s).
```

Le troisième test a une fois de plus confirmé l'écart de performances entre les algorithmes Welsh-Powell et Hill-Climbing, que ce soit au niveau du temps d'exécution ou du nombre de couleurs utilisées.

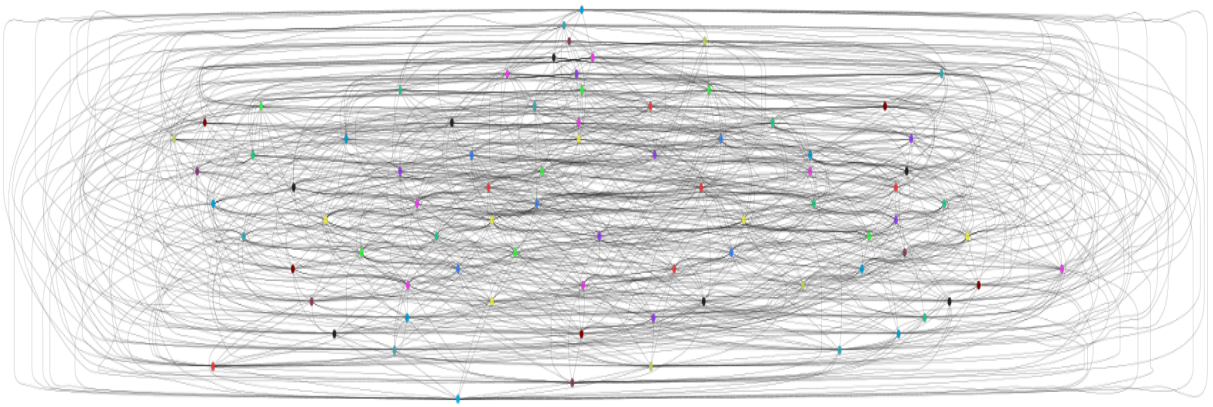
Welsh-Powell a démontré une grande efficacité en exécutant l'algorithme en seulement 0,043 secondes. Hill-Climbing, quant à lui, a mis plus de 1,38 secondes, soit un temps 32 fois supérieur à celui de Welsh-Powell..

Welsh-Powell a produit une solution nécessitant 11 couleurs, un résultat raisonnablement optimisé. Hill-Climbing, en revanche, a utilisé 60 couleurs, un écart impressionnant qui illustre un manque d'efficacité dans la minimisation des ressources.

4. Quatrième test : queen9_9.col

```
Nom du fichier à charger : queen9_9.col
Graphe lu: 81 nœuds, 2112 arêtes
Graphe chargé : 81 noeuds, 2112 arêtes
```

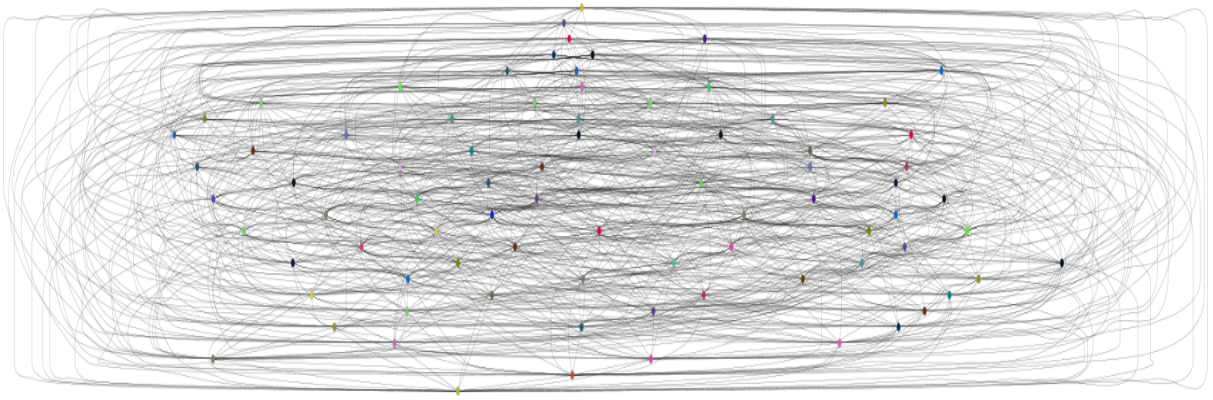
Coloration avec l'algorithme Welsh-Powell :



```
Statistiques:
Temps d'exécution: 0.0926625729 secondes
Nombre d'itérations: 1
Conflits finaux: 0
```

```
Votre choix : 12
La coloration est valide et utilise 15 couleur(s).
```

Coloration avec l'algorithme Hill-Climbing



```
Statistiques:
Temps d'exécution: 1.2261998653 secondes
Nombre d'itérations: 1
Conflits finaux: 0
```

```
Votre choix : 12
La coloration est valide et utilise 33 couleur(s).
```

Lors de ce dernier test, les résultats obtenus ont une nouvelle fois mis en évidence les différences significatives entre les algorithmes Welsh-Powell et Hill-Climbing, aussi bien en termes de temps d'exécution que du nombre de couleurs utilisées.

Welsh-Powell a terminé son exécution en 0,092 secondes, conservant une rapidité remarquable malgré une légère augmentation par rapport aux tests précédents.
Hill-Climbing, en revanche, a nécessité 1,22 seconde, soit environ 13 fois plus de temps que Welsh-Powell.

Ce résultat souligne une nouvelle fois l'avantage structuré de Welsh-Powell, qui reste rapide même sur des graphes de plus grande complexité. Hill-Climbing, bien qu'exploratoire, montre une lenteur notable dans ses itérations et sa convergence.

Welsh-Powell a produit une solution utilisant 15 couleurs, un résultat satisfaisant pour un graphe plus complexe. Hill-Climbing, de son côté, a nécessité 33 couleurs, soit plus du double de celles utilisées par Welsh-Powell.

- 5.2.2 Discussion des résultats obtenus

  À travers les quatre tests effectués, Welsh-Powell s'est imposé comme l'algorithme le plus efficace pour résoudre les problèmes de coloration de graphes. Il allie une rapidité d'exécution à une capacité à minimiser les couleurs utilisées, le rendant particulièrement adapté à des applications réelles.
  Hill-Climbing, en revanche, bien qu'intéressant sur le plan heuristique, nécessite des améliorations pour rivaliser avec Welsh-Powell, notamment en termes de temps d'exécution et d'optimisation des solutions produites.

  Ces résultats soulignent également l'importance de choisir l'algorithme en fonction des besoins spécifiques d'une application, en tenant compte des compromis entre rapidité, qualité et flexibilité.

6. Ressources et Références

- 6.1 Bibliothèques logicielles utilisées
  - `from graphviz import Graph, Digraph` :
    La bibliothèque **Graphviz** est utilisée pour la **visualisation** des graphes. Elle permet de générer des graphiques à partir d'une description textuelle des graphes sous la forme de fichiers DOT. Vous pouvez ainsi créer et visualiser des graphes directs (**Digraph**) ou non directs (**Graph**).
  - `import random` :
    La bibliothèque **random** est utilisée pour générer des éléments aléatoires. Elle peut être utilisée, par exemple, pour générer des graphes aléatoires, ou pour ajouter un aspect aléatoire dans les algorithmes, comme dans le **Hill-Climbing**.
  - `import copy` :
    La bibliothèque **copy** est utilisée pour effectuer des copies d'objets en Python. Cela est utile, par exemple, pour dupliquer des graphes ou des solutions avant de les manipuler sans affecter l'original. Les fonctions comme `copy.deepcopy()` permettent de créer des copies indépendantes de structures complexes.
  - `from PIL import Image` :
    PIL (Python Imaging Library) est utilisée pour la manipulation des images. Dans le contexte de votre projet, elle pourrait être utilisée pour charger, afficher, ou enregistrer des images de graphes ou d'autres résultats sous forme d'images (par exemple, après avoir généré des visualisations de graphes avec Graphviz).

7. Conclusion

7.1 Synthèse des travaux réalisés

Ce projet a été une opportunité enrichissante pour approfondir nos connaissances théoriques et pratiques sur les graphes. Il s'est structuré autour de plusieurs objectifs qui nous ont permis de progresser à la fois techniquement et personnellement.

La première étape du projet a été consacrée à l'étude de la problématique de coloration. Nous avons découvert que la coloration des graphes est un problème algorithmique fondamental avec des applications variées, notamment en planification, allocation de ressources et cartographie. Cette exploration nous a permis de comprendre les concepts théoriques de base : le nombre chromatique, la coloration propre, et les propriétés des graphes. Ainsi que les limitations inhérentes au problème et donc les défis qu'il pose en matière de conception d'algorithmes efficaces.

La seconde phase du projet consistait à développer plusieurs algorithmes pour résoudre le problème de coloration. Cela nous a permis de comparer différentes approches, notamment les algorithmes gloutons. Ils sont simples et rapides, ils affectent les couleurs aux sommets en suivant un ordre donné. Mais aussi les approches heuristiques et probabilistes telles que l'algorithme de Welsh-Powell ou des algorithmes inspirés de la théorie des graphes aléatoires.

Cette phase nous a enseigné l'importance d'adapter les outils en fonction des contraintes spécifiques (efficacité, optimalité) et de tester rigoureusement nos implémentations. Nous avons appris à analyser la complexité temporelle et spatiale des algorithmes, ainsi qu'à évaluer leurs performances sur des jeux de données variés.

En parallèle, nous avons conçu des outils pour visualiser et valider les colorations obtenues. Cela nous a conduit à utiliser des bibliothèques comme Graphviz pour générer des graphes et représenter graphiquement les résultats. Cette partie nous a permis de développer une approche méthodique dans la conception logicielle, en structurant notre code autour de classes et de fonctions réutilisables. Puis, de comprendre l'importance de l'interface utilisateur dans les outils de visualisation, en nous assurant que les résultats étaient clairs et interprétables.

Nous avons amélioré nos compétences en algorithmique avancée, en structures de données et en programmation Python. Nous avons également gagné en aisance dans l'utilisation d'outils de visualisation et dans l'analyse de résultats expérimentaux.

Ce projet nous a appris à organiser notre travail de manière itérative, en alternant entre phases de conception, développement et évaluation. Nous avons pris conscience de l'importance de la documentation et des tests pour assurer la robustesse et la clarté de nos implémentations.

Une part importante du projet a consisté à faire des recherches bibliographiques pour comprendre les méthodes existantes et nous en inspirer. Cette démarche nous

a sensibilisé à l’importance de la veille technologique et scientifique dans un domaine en constante évolution.

● 7.2 Propositions d'amélioration

Les améliorations futures pourraient inclure l’intégration de nouvelles techniques de résolution et l’optimisation des algorithmes existants. Cela pourrait être en développant les autres algorithmes présentés dans l’énoncé et en classe pour le problème de coloration de graphe.

● 7.3 Perspectives pour de futures recherches

Des recherches complémentaires pourraient explorer l’application des algorithmes à des contextes professionnels. Une autre perspective intéressante serait d'étudier les graphes dynamiques, où les sommets ou les arêtes évoluent dans le temps, afin de développer des solutions adaptées à des systèmes en constante évolution.