Coloriage de graphes
Une application à la résolution du Sudoku

Algorithmique des graphes

HANNIZ Anis, BEN HAMMOU Eddine, BITHO Joshua

UNIVERSITÉ DE ROUEN NORMANDIE

Rapport du Projet : Coloriage des graphes appliqué à la résolution du Sudoku.

9 Janvier 2024

Contenu

1 Introduction **2**
- 1.1 Origines de la Coloration des Graphes . . . 2
- 1.2 Importance et Applications Contemporaines . . . 2

2 Structure du Projet **3**
- 2.1 Fichiers Python . . . 3
- 2.2 Ressources et Documentation . . . 3

3 Types de Données Utilisés dans le Projet **4**
- 3.1 Graphes . . . 4
- 3.2 Dictionnaires et Ensembles . . . 4
- 3.3 Listes et Tuples . . . 4
- 3.4 Interfaces Utilisateur et Interactions . . . 4
- 3.5 Grilles de Sudoku . . . 4

4 Pseudocode des Algorithmes de Coloration de Graphes **5**
- 4.1 Coloriage Glouton . . . 5
- 4.2 Coloriage Welsh-Powell . . . 5
- 4.3 Coloriage Welsh-Powell Dynamique . . . 6
- 4.4 Coloriage par Backtracking . . . 6

5 Complexité des Algorithmes **7**
- 5.1 Complexité du Coloriage Glouton . . . 7
- 5.2 Complexité du Coloriage Welsh-Powell . . . 7
- 5.3 Complexité du Coloriage Welsh-Powell Dynamique . . . 7
- 5.4 Complexité du Coloriage par Backtracking . . . 7

6 Résolution de Sudoku **8**
- 6.1 `print_sudoku` : Affichage du Sudoku . . . 8
- 6.2 `is_valid` : Validation des Mouvements . . . 8
- 6.3 `find_empty_location` : Recherche de Cellules Vides . . . 8

6.4 `solve_sudoku` : Algorithme de Résolution 8
6.5 `solve_sudoku_board` : Interface de Résolution 8

7 Interface Utilisateur et Interaction **9**
7.1 Interface Graphique 9
7.2 Gestion de Graphes 9
7.3 Résolution de Sudoku 9
7.4 Interaction et Flux de Travail 9
7.5 Détails Techniques 9

8 Conclusion **10**

1 Introduction

Ce rapport présente une exploration détaillée d'une application Python conçue pour résoudre deux problématiques complexes et interconnectées : la coloration de graphes et la résolution de Sudoku. L'approche adoptée pour relever ces défis s'appuie sur la bibliothèque NetworkX, qui facilite la manipulation des structures de graphes, et Matplotlib, qui permet une visualisation graphique riche et intuitive. Le projet se compose de trois fichiers Python principaux — `main.py`, `graphe_coloration.py` et `sudoku_solver.py` — ainsi que d'un fichier texte (`sudo.txt`) servant de source pour les grilles de Sudoku.

La genèse de la coloration des graphes, un domaine mathématique profondément enraciné dans l'histoire, est brièvement retracée pour contextualiser son importance et son application contemporaine.

1.1 Origines de la Coloration des Graphes

La coloration des graphes, initialement conçue pour résoudre des problèmes de coloration cartographique, remonte à la conjecture des quatre couleurs posée par Francis Guthrie en 1852. Cette conjecture postulait que quatre couleurs suffisaient pour colorer n'importe quelle carte de manière à ce que deux régions adjacentes ne partagent jamais la même couleur. Bien que la preuve initiale proposée par Alfred Kempe ait été ultérieurement réfutée, elle a ouvert la voie à des avancées significatives, telles que le théorème des cinq couleurs formulé par Percy John Heawood.

L'histoire du domaine est marquée par des recherches incessantes pour valider la conjecture des quatre couleurs, qui n'ont abouti qu'avec les travaux de Kenneth Appel et Wolfgang Haken au XXe siècle. Leur succès a non seulement confirmé la conjecture mais a aussi signalé l'entrée des ordinateurs comme outils de preuve dans les mathématiques formelles, une véritable révolution dans les méthodes de validation mathématique.

1.2 Importance et Applications Contemporaines

De nos jours, la coloration des graphes dépasse largement le cadre des cartes géographiques pour s'étendre à de nombreux autres champs, tels que l'optimisation de réseaux, la planification et l'algorithmique dans divers contextes opérationnels. Elle constitue un outil essentiel pour aborder des problèmes variés, allant de l'organisation d'horaires à la répartition de ressources, en passant par les défis logistiques de la vie quotidienne et industrielle.

Ce rapport aborde ces sujets non seulement d'un point de vue théorique mais aussi pratique, en présentant une application concrète de ces concepts à travers le développement et la mise en œuvre d'algorithmes sophistiqués dans un programme Python.

2 Structure du Projet

Le projet est structuré en plusieurs fichiers Python et autres ressources, chacun ayant un rôle spécifique dans l'application. Voici une description des éléments clés de la structure du projet :

2.1 Fichiers Python

- `main.py` : Ce fichier sert de point d'entrée pour l'application. Il initialise l'interface utilisateur et gère l'interaction de l'utilisateur avec les fonctionnalités du programme.
- `graphe_coloration.py` : Contient les algorithmes de coloration de graphe, y compris les implémentations des stratégies gloutonne, de Welsh-Powell, et de backtracking.
- `sudoku_solver.py` : Implémente l'algorithme de résolution de Sudoku, y compris la recherche de cellules vides, la validation des mouvements et l'algorithme de backtracking pour résoudre le Sudoku.

2.2 Ressources et Documentation

- `README.md` : Fournit des informations générales sur le projet, y compris la façon de l'exécuter, les dépendances nécessaires, et une description générale des fonctionnalités.
- `sudo.txt` : Un exemple de grille de Sudoku au format texte, utilisé pour charger rapidement une grille dans l'application et tester l'algorithme de résolution.

Chaque fichier est essentiel au fonctionnement global de l'application et contribue à une certaine fonctionnalité ou à une partie de l'interface utilisateur.

3 Types de Données Utilisés dans le Projet

Dans ce projet, plusieurs types de données sont utilisés pour manipuler et résoudre les problèmes de coloration de graphes et de Sudoku. Les types de données principaux sont décrits ci-dessous pour fournir une compréhension claire de leur utilisation et de leur importance dans l'application.

3.1 Graphes

Le type de données central pour la coloration de graphes est le **graphe**, représenté par la structure de données `Graph` de la bibliothèque NetworkX. Les graphes sont composés de nœuds et d'arêtes, où:

- Les **nœuds** représentent les entités dans le problème de coloration, et chaque nœud est identifié par un identifiant unique.
- Les **arêtes** représentent les relations ou les liens entre les nœuds. Dans le contexte de la coloration de graphes, deux nœuds reliés par une arête ne peuvent pas partager la même couleur.

3.2 Dictionnaires et Ensembles

Pour stocker les couleurs attribuées aux nœuds, nous utilisons un **dictionnaire** où la clé est le nœud et la valeur est la couleur assignée. Les couleurs sont représentées par des entiers uniques.

Les **ensembles** sont utilisés pour maintenir une collection non ordonnée de couleurs déjà utilisées par les nœuds voisins lors de la tentative d'attribution d'une nouvelle couleur à un nœud.

3.3 Listes et Tuples

Les **listes** sont utilisées pour stocker des collections ordonnées, telles que les nœuds triés par degré ou les valeurs d'une grille de Sudoku.

Les **tuples** sont des collections immuables qui sont utilisées pour stocker les paires de nœuds (arêtes) dans le graphe.

3.4 Interfaces Utilisateur et Interactions

L'interface utilisateur est construite en utilisant le module `tkinter`, qui fournit des widgets tels que des boutons, des zones de texte, et des menus déroulants. Les interactions de l'utilisateur avec ces widgets sont gérées par des événements qui déclenchent des fonctions correspondantes dans le code.

3.5 Grilles de Sudoku

Une grille de Sudoku est représentée par une **liste de listes**, où chaque sous-liste représente une ligne de la grille. Les cellules vides sont indiquées par un 0, et les chiffres de 1 à 9 sont utilisés pour remplir la grille.

4 Pseudocode des Algorithmes de Coloration de Graphes

Cette section fournit une représentation en pseudocode des algorithmes implémentés dans le fichier `graphe_coloration.py`. Chaque algorithme est décrit étape par étape pour mettre en évidence sa logique algorithmique.

4.1 Coloriage Glouton

```
Fonction greedy_coloring(graph):
    trier les nœuds de graph par degré décroissant
    pour chaque nœud dans les nœuds triés:
        initialiser un ensemble de couleurs des voisins
        pour chaque voisin du nœud:
            si le voisin a une couleur:
                ajouter la couleur du voisin à l'ensemble
        initialiser color à 0
        tant que color est dans l'ensemble des couleurs des voisins:
            color = color + 1
        attribuer color au nœud
    retourner le dictionnaire des couleurs
```

4.2 Coloriage Welsh-Powell

```
Fonction welsh_powell_coloring(graph):
    calculer le degré de chaque sommet
    trier les sommets par degré dans l'ordre décroissant
    initialiser les couleurs à vide
    pour chaque sommet dans l'ordre trié:
        si le sommet n'a pas encore de couleur:
            trouver la première couleur non utilisée par les voisins
            attribuer cette couleur au sommet
    retourner le dictionnaire des couleurs et le nombre chromatique
```

4.3 Coloriage Welsh-Powell Dynamique

```
Fonction welsh_powell_coloring_dynamic(graph):
    Fonction interne welsh_powell_coloring_internal(graph):
        trier les nœuds par nombre de voisins dans l'ordre décroissant
        colorier chaque nœud en évitant les couleurs des voisins
        retourner le dictionnaire des couleurs
    initialiser la coloration avec welsh_powell_coloring_internal(graph)
    Fonction update_coloring(event):
        mettre à jour le graphe en fonction de l'événement
        recalculer la coloration
        retourner la nouvelle coloration
    retourner la coloration et la fonction update_coloring
```

4.4 Coloriage par Backtracking

```
Fonction backtrack_coloring(graph):
    Fonction is_safe(node, color, colors):
        pour chaque voisin du nœud:
            si le voisin a la couleur en question:
                retourner faux
        retourner vrai

    Fonction backtrack(nodes, colors):
        si il n'y a plus de nœuds à colorier:
            retourner vrai et les couleurs
        sélectionner un nœud non coloré
        pour chaque couleur possible:
            si la couleur est sûre:
                attribuer la couleur au nœud
                si backtrack est vrai pour le reste:
                    retourner vrai et les couleurs
                sinon:
                    retirer la couleur du nœud
        retourner faux

    initialiser les nœuds et les couleurs
    appeler la fonction backtrack avec tous les nœuds et les couleurs
    si la coloration est réussie:
        retourner le résultat
    sinon:
        lever une exception
```

5 Complexité des Algorithmes

La complexité des algorithmes de coloration de graphes est un aspect crucial pour comprendre leur efficacité et leur applicabilité à des graphes de différentes tailles et densités.

5.1 Complexité du Coloriage Glouton

L'algorithme de coloriage glouton a une complexité temporelle de $O(n + m)$ où n est le nombre de nœuds et m est le nombre d'arêtes. Cela est dû au tri des nœuds par degré, qui peut être effectué en $O(n \log n)$ et à l'itération sur chaque nœud et ses voisins pour déterminer la couleur minimale non utilisée.

5.2 Complexité du Coloriage Welsh-Powell

Le coloriage Welsh-Powell a également une complexité temporelle de $O(n^2)$ dans le pire des cas, car il nécessite un tri des sommets par degré suivi d'une vérification des couleurs des voisins pour chaque sommet, ce qui peut nécessiter une inspection de tous les autres sommets dans le cas d'un graphe dense.

5.3 Complexité du Coloriage Welsh-Powell Dynamique

La complexité du coloriage Welsh-Powell dynamique dépend de la complexité de l'algorithme Welsh-Powell initial plus la complexité des mises à jour dynamiques. Chaque mise à jour est effectuée en $O(d)$ où d est le degré du nœud ajouté ou supprimé. Ainsi, la complexité totale devient $O(n^2) + kO(d)$ pour k mises à jour.

5.4 Complexité du Coloriage par Backtracking

Le coloriage par backtracking a une complexité temporelle exponentielle dans le pire des cas, car il explore potentiellement tous les arrangements possibles de couleurs. En termes de notation O, cela est souvent exprimé comme $O(k^n)$, où k est le nombre de couleurs disponibles et n le nombre de nœuds. C'est l'un des algorithmes les plus coûteux en termes de temps de calcul, mais il trouve une solution garantie si elle existe.

Ces complexités nous permettent de choisir un algorithme adapté à la taille et à la structure spécifique du graphe que nous cherchons à colorer. Pour les grands graphes, les méthodes heuristiques telles que le coloriage glouton ou Welsh-Powell sont généralement préférées, tandis que le backtracking peut être réservé à des graphes de petite taille ou lorsque une coloration exacte est nécessaire.

6 Résolution de Sudoku

Cette section se concentre sur les fonctions Python utilisées pour résoudre un Sudoku. Le Sudoku est un jeu de logique populaire où l'objectif est de remplir une grille 9x9 de sorte que chaque ligne, chaque colonne et chaque sous-grille 3x3 contiennent tous les chiffres de 1 à 9.

6.1 `print_sudoku` : Affichage du Sudoku

La fonction `print_sudoku` est utilisée pour afficher l'état actuel de la grille de Sudoku en output. Elle parcourt chaque ligne de la grille et affiche les valeurs dans un format lisible. Mais nous avons choisi de ne pas l'utiliser par la suite car elle a servi pour les premiers essais.

6.2 `is_valid` : Validation des Mouvements

La fonction `is_valid` vérifie si un mouvement est valide. Elle s'assure qu'un chiffre spécifique peut être placé dans une cellule donnée sans violer les règles du Sudoku. Cette vérification est effectuée en s'assurant que le chiffre n'est pas déjà présent dans la même ligne, la même colonne, ou la même sous-grille 3x3.

6.3 `find_empty_location` : Recherche de Cellules Vides

La fonction `find_empty_location` cherche une cellule vide dans la grille, ce qui est nécessaire pour les étapes suivantes du processus de résolution. Si une cellule vide est trouvée, ses coordonnées sont retournées.

6.4 `solve_sudoku` : Algorithme de Résolution

L'algorithme principal pour résoudre le Sudoku est implémenté dans la fonction `solve_sudoku`. Cette fonction utilise une approche récursive et de backtracking. Elle cherche d'abord une cellule vide, puis teste des chiffres valides dans cette cellule. Si un chiffre valide conduit à une solution, la fonction continue; sinon, elle annule (backtrack) et essaie un autre chiffre.

6.5 `solve_sudoku_board` : Interface de Résolution

La fonction `solve_sudoku_board` sert d'interface pour démarrer le processus de résolution. Elle appelle `solve_sudoku` et retourne le résultat, indiquant si une solution a été trouvée, et le cas échéant, l'état résolu de la grille.

Cet ensemble de fonctions constitue une implémentation complète et efficace pour résoudre des Sudoku de différentes complexités.

7 Interface Utilisateur et Interaction

Le fichier `main.py` est le cœur de l'interface utilisateur de notre application, construite en utilisant la bibliothèque `tkinter`. Ce module orchestre l'interaction de l'utilisateur avec les fonctionnalités de coloration de graphes et de résolution de Sudoku.

7.1 Interface Graphique

L'interface graphique principale est initialisée en créant une instance de la classe `Tk`. Cette fenêtre sert de conteneur pour tous les widgets, tels que les boutons, les menus déroulants et les zones de saisie, permettant à l'utilisateur de naviguer dans l'application.

7.2 Gestion de Graphes

Des fonctions comme `add_vertex`, `remove_vertex`, et `refresh_graph` sont reliées à des boutons pour permettre à l'utilisateur d'interagir dynamiquement avec les graphes. La coloration est mise à jour en temps réel grâce à des fonctions telles que `draw_graph2` qui utilisent la bibliothèque `matplotlib` pour le rendu visuel.

7.3 Résolution de Sudoku

Une interface utilisateur distincte est fournie pour la résolution de Sudoku via la fonction `sudoku_ui`. Elle permet aux utilisateurs de saisir les chiffres d'un Sudoku, de résoudre la grille, d'importer des grilles depuis un fichier texte ou de réinitialiser la grille à son état vide.

7.4 Interaction et Flux de Travail

Les interactions de l'utilisateur avec l'interface sont gérées par des variables globales et des fonctions de rappel (*callbacks*). Lorsque l'utilisateur sélectionne une action ou un algorithme via l'interface, des fonctions correspondantes telles que `color_graph_ui` sont appelées pour traiter la demande et mettre à jour l'interface avec les résultats.

7.5 Détails Techniques

Le placement et le centrage des fenêtres sont gérés par `center_window`, qui calcule la position optimale pour la fenêtre principale et les fenêtres auxiliaires. Les fonctions de coloration de graphe et la résolution de Sudoku sont importées depuis les modules `sudoku_solver` et `graphe_coloration` respectivement, démontrant une séparation claire de l'interface utilisateur et de la logique métier.

8 Conclusion

En résumé, ce projet a démontré l'efficacité et la polyvalence des algorithmes de coloration de graphes et de résolution de Sudoku. À travers l'implémentation de différentes stratégies algorithmiques et l'utilisation d'une interface utilisateur intuitive, nous avons pu aborder des problèmes complexes de manière visuelle et interactive.
Les méthodes de coloration de graphes, allant du glouton au backtracking, révèlent la profondeur et la complexité de la théorie des graphes. De plus, l'application de ces méthodes à la résolution de Sudoku illustre la manière dont des concepts mathématiques abstraits peuvent être appliqués à des jeux de logique populaires, rendant les mathématiques à la fois accessibles et divertissantes.

L'utilisation de l'interface graphique construite avec `tkinter` a permis une manipulation et une visualisation directes des graphes, rendant l'expérience utilisateur plus enrichissante. L'interaction entre l'interface utilisateur et les algorithmes de coloration de graphes s'est avérée fluide, permettant aux utilisateurs de tester différentes stratégies de coloration en temps réel.

L'approche modulaire adoptée pour la structure du projet a facilité la maintenance et l'expansion éventuelle de l'application. En gardant la logique métier séparée de l'interface utilisateur, nous avons créé un système flexible qui peut être étendu pour inclure de nouvelles fonctionnalités ou optimisations d'algorithmes.

En conclusion, ce projet ne se limite pas à un simple exercice académique ; il établit un pont entre la théorie et la pratique, offrant des outils qui peuvent être adaptés à une variété de défis du monde réel. Il s'agit d'un exemple frappant de la manière dont l'informatique théorique peut être rendue concrète, fournissant des solutions tangibles tout en restant fidèle aux fondements rigoureux des mathématiques.