Problème de coloration de graphe et hyper-heuristique

constance.bau

February 2025

Contents

1 Description du sujet

L'objectif principal de ce projet est de résoudre le problème de coloration de graphe avec 4 heuristiques simples différentes et avec une hyper-heuristique utilisant ces 4 heuristiques simples.

Commençons par définir ce qu'est une hyper-heuristique. Un hyper-heuristique est "un algorithme (à un niveau supérieur) qui "sélectionne" des heuristiques appropriées (à un niveau inférieur) pour les appliquer aux problèmes en question. Une hyper-heuristique se concentre sur l'exploration d'un espace de recherche d'heuristiques plutôt que sur la recherche directe de solutions aux problèmes" ((Burke et al., 2007)).

Pour voir cela en pratique, nous allons nous intéresser au problème de coloration de graphe, un problème simple à comprendre mais difficile à résoudre de manière exacte car il est np difficile. Nous implémenterons 4 heuristiques simples pour résoudre ce problème sur différentes instances trouvées sur des sites comme DICMACS puis nous implémenterons une hyperheuristique, c'est à dire une méthode que l'on va entraîner sur plusieurs graphe pour qu'elle soit ensuite capable de trouver la meilleure combinaison d'heuristiques simples à appliquer au problème à tour de rôle pour améliorer la résolution du problème.

L'idée derrière une hyper heuristique ressemble au méta apprentissage, en effet, l'hyperheuristique s'entraîne sur plusieurs instances pour trouver une combinaison d'heuristiques qui soit la meilleure possible pour résoudre le problème. Ensuite cette combinaison d'heuristiques devraient être très bonne pour résoudre le problème de coloration de graphes sur tous les types de graphes. L'idée nous est venue de cet article (Burke et al., 2007) qui présente une hyper-heuristique basée sur les graphes pour les problèmes de planification des emplois du temps éducatifs. Ayant retenu l'idée générale de leur hyper heuristique nous avons essayer d'adapter cette idée ainsi que les heuristiques simples pour le problème de coloration de graphe. Comme les codes de l'article n'était pas fourni et qu'il y avait peu de pseudo code dans l'article mais plutot des explications générale, nous avons improvisé l'implémentation ce qui fait que l'on s'est un peu éloigné et que l'on a simplifié cette méthode. Nous avons aussi utilisée l'article (Aslan and Baykan, 2016) pour avoir une idée de différentes heuristiques simples pour le problème du coloriage de graphe.

2 Description des méthodes choisies

Nous présenterons d'abord les 4 heuristiques simples pour la coloration de graphe que nous avons trouvées dans (Aslan and Baykan, 2016), puis, après avoir parlé de façon générale des hyper-heuristiques, nous expliquerons celle que nous avons choisie.

2.0.1 FirstFit

Cette heuristique se résume par la phrase : "la première couleur possible est choisie".

Voici les principales étapes de l'algorithme :

- Extraire la liste des nœuds du graphe qui n'ont pas encore été coloriés.
- Pour chaque nœud sélectionné :
  - Récupérer les couleurs utilisées par ses voisins
  - Parcourir dans l'ordre croissant toutes les couleurs déjà utilisées dans le graphe et colorier le noeud avec la première qui n'appartient à l'ensemble des couleurs voisines du noeud. Si aucune n'est trouvée, colorier le noeud avec une nouvelle couleur.

2.0.2 WelshPowell

Cet algorithme consiste à trier les noeuds par degrés décroissants et chercher à colorier d'abord ceux avec le plus de voisins. De plus, après avoir colorier un noeud, l'algorithme colorie le maximum de noeuds non adjacents à ce noeud de la même couleur.

Etape par étape, cela donne :

1. L'algorithme trie les nœuds par ordre décroissant de degré.
2. Le premier nœud non colorié est colorié avec la plus petite couleur disponible.
3. Chaque noeud non-adjacent est testé et colorié avec la même couleur si c'est possible.
4. On revient à l'étape 2.

2.0.3 smallest Degree Ordering Algorithm (SDO)

C'est le même principe (ou le principe inverse en fonction de la manière dont on voit les choses) que l'algorithme WelshPowell sauf que les noeuds sont triés par ordre croissants de nombre de voisins.

2.0.4 Degree of Saturation Algorithm (DSATUR)

Cette méthode se concentre davantage sur le degré de saturation des noeuds, c'est à dire le nombre de couleurs voisines différentes.
En voici les principales étapes :

1. Les noeuds sont triés par degrés de saturation décroissants
2. Le nœud avec la saturation minimale est colorié avec la plus petite couleur disponible.
3. Les degré de saturation des noeuds voisins sont mis à jour.
4. Les noeuds restants sont à nouveau triès par degrés de saturation et l'on retourne à l'étape 2.

2.0.5 Hyperheuristique

Dans cette partie, nous évoquerons les hyper-heuristiques dans le cadre général puis nous verrons en détails l'hyper-heuristique que nous avons implémenté, inspirée de celle développée dans (Burke et al., 2007).

Le développement des hyper-heuristiques est motivé par l'objectif d'accroître le niveau de généralité pour résoudre automatiquement une gamme de problèmes. Une hyper-heuristique peut être vue comme un algorithme (à un niveau supérieur) qui "sélectionne" des heuristiques appropriées (à un niveau inférieur) pour les appliquer aux problèmes en question. Une hyper-heuristique se concentre sur l'exploration d'un espace de recherche d'heuristiques plutôt que sur la recherche directe de solutions aux problèmes ((Burke et al., 2007)). On pourrait dire que les hyper heuristiques sont aux méthodes d'optimisation ce que le méta apprentissage est à l'intelligence artificielle. Présentons les hyper-heuristiques basée sur une recherche tabou.

Initialement, la recherche tabou est une méta heuristique qui cherche la meilleure solution à un problème en explorant l'espace de solution du problème en tentant de se diriger vers les meilleures solutions. Une liste tabou des dernières solutions visitées est créée afin d'éviter de revisiter des solution venant d'être visitées et de tomber dans des extremums locaux. On peut "transformer" cette méthode en hyper heuristique de la manière suivante : la recherche ne se fait plus sur l'espace des solutions d'un problème mais sur l'espace des heuristiques d'une classe/gamme de problèmes. Ainsi on recherche par exemple la combinaison d'heuristiques donnant les meilleures solutions à un ensemble de problèmes. Ensuite, on utilise cette combianaison pour résoudre d'autres problèmes de cette classe. Par exemple, dans l'article (Burke et al., 2007) l'hyper heuristique s'adapte aux différents problèmes d'optimisation des emplois du temps et de l'affectation des ressources. Après l'avoir entraîner sur un ensemble de problème on peut l'utiliser pour résoudre n'importe quel problème de cette famille de problème rapidement.

L'avantage des hyper-heuristiques ne réside pas dans le fait d'améliorer la qualité de la solution trouvée à une instance particulière d'un problème mais plutôt dans le fait de trouver une méthode rapide que l'on pourra appliquer sur toutes les instances d'une classe de problème pour obtenir rapidement une solution quasiment aussi bonne que la meilleure existante, sans modifier la méthode en fonction des spécificités de l'instances. La méthode doit s'adapter et donner de bons résultats sur une large gamme de problème.

3 Description de l'application choisie

Nous avons choisi une classe de problème simple : les problèmes de coloration de graphes. Ce problème vise à trouver le nombre minimum de couleur à utiliser pour colorier les sommets d'un graphe de telle sorte que deux sommets reliés par une arrête n'aient pas la même couleur. Le problème de coloration de graphe est NP-complet car il appartient à NP (une solution peut être vérifiée en temps

polynomial) et il est NP-difficile (car un problème NP-complet comme SAT peut être réduit à un problème de coloration en temps polynomial).

En fonction des spécificités du graphe à colorier (densité, nombre de voisins maximum d'un sommet, rapport sommets/arrêtes, homogénéité,...), les méthodes permettant d'obtenir la meilleure solution le plus rapidement varient, d'où l'idée de créer une hyper heuristiques donnant des résultats proches de l'optimale pour tous les types de graphes. En pratique, ce problème peut permettre de résoudre des problèmes d'affectation de tâches. Imaginons que l'on cherche à affecter différents cours dans des salles de classes et que l'on cherche à utiliser le moins de salles de classe possibles. Mettons ce problème sous la forme d'un problème de coloration de graphe : chaque noeuds du graphe représente un cours, chaque couleur représente une salle et une arrête entre deux noeuds signifie que les deux cours représentés par ces noeuds ont lieu en même temps. Dans ce cas, résoudre le problème de coloration sur ce graphe permet de trouver le minimum de salles nécessaires et de quelle manière placer les différents cours dans les différentes salles.

4 Explications sur la façon dont la méthode s'applique à l'application

Voyons maintenant de quelle manière appliquer une hyper heuristique par recherche Tabou à notre classe de problème. Tout d'abord il faut que l'hyper heuristique nous obtienne la meilleure combinaison d'heuristiques puis nous utilisons cette combinaison pour colorier différentes instances de graphes.

Dans notre cas, l'espace de recherche va être constitué des séquences avec répétitions des 4 heuristiques simples présentées plus haut. On choisi les séquences de taille 8 (choix arbitraire pour que l'espace de recherche ne soit pas trop grand mais bien sûr plus la taille de la séquence est grande plus l'on a de possibilités et donc on peut davantage améliorer la qualité du résultat). L'espace de recherche est donc de taille $4^8 = 65536$.

Par exemple, la séquence [sdo, sdo, firstfit, dsatur, WelshPowell, WellPowell, dsatur, dsatur] est un élément de l'espace de recherche.

Voyons les différentes étape de l'adaptation de cette méthode à la coloration de graphe :

1. on donne à l'algorithme un ensemble de graphes servant d'ensemble d'entraînement pour l'hyperheuristique
2. les graphes sont pris un par un et les étapes suivantes sont répétées sur chaque graphes :
   (a) un tirage aléatoire avec remise de 8 heuristiques est effectué pour obtenir une séquence d'heuristiques.
   (b) le graphe est colorié avec cette séquence d'heuristiques. Voyons un exemple pour mieux comprendre. Si par exemple le graphe contient

64 noeuds et que la séquence d'heuristiques est [sdo, sdo, firstfit, dsatur, WelshPowell, WellPowell, dsatur, dsatur] alors 9 noeuds seront coloriés par sdo, puis à nouveau 9 noeuds parmi les non coloriés restants par sdo, puis 9 noeuds parmi les restants seront coloriés par firstfit,... Si le nombre de noeuds n'est pas un multiple de 8 alors la dernière heuristique colorie le résultats de la division euclidienne du nombre de noeuds par 8 plus le reste. Ensuite on retient le nombre de couleur utilisées pour le coloriage, on stocke la séquence d'heuristiques dans la liste Tabu et on la définit comme meilleure solution. On modifie ensuite légèrement la séquence actuelle (par exemple, à 2 indices aléatoires de la séquence, les 2 heuristiques actuelles sont remplacées par 2 heuristiques aléatoires. Puis on efface le coloriage du graphe et on le colorie à nouveau avec la nouvelle séquence. Si le nombre de couleur est inférieur, la nouvelle séquance est définie comme meilleure solution et, dans tous les cas, elle est ajoutée à la liste Tabu. On continue durant un certain nombre d'itérations.

3. Après avoir fait cela sur chaque graphe on obtient un ensemble de séquences d'heuristiques qui sont les meilleures séquences d'heuristiques de chaque graphe.

4. Pour chaque position dans la séquence, on regarde quelle est l'heuristique la plus fréquente à cette position parmi l'ensemble des meilleures solutions de chaque graphe et on met cette heuristique à cette position dans la meilleure solution globale.

5. On peut ensuite colorier n'importe quelle instance du problème de coloration de graphe avec la séquence d'heuristiques formant dans la meilleure solution globale.

5 Modélisation UML générique commentée

5.1 Diagramme de classes avant implémentation

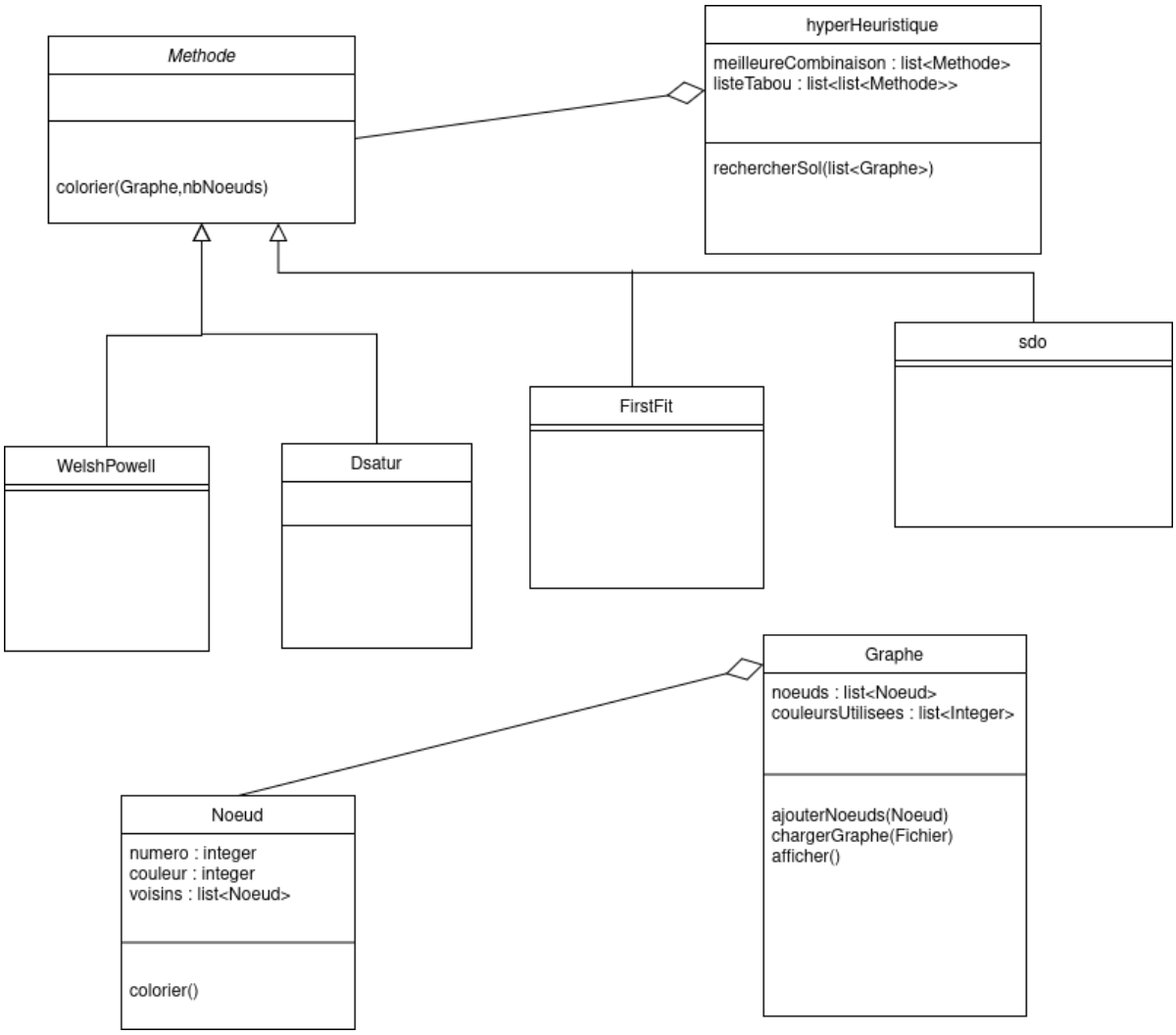

Figure 1: diagramme de classes avant implémentation

5.2 Diagramme de classes ajusté après implémentation

Les méthodes que j'ai rajoutées au fur et à mesure de l'implémentation ont été mises en rouge.

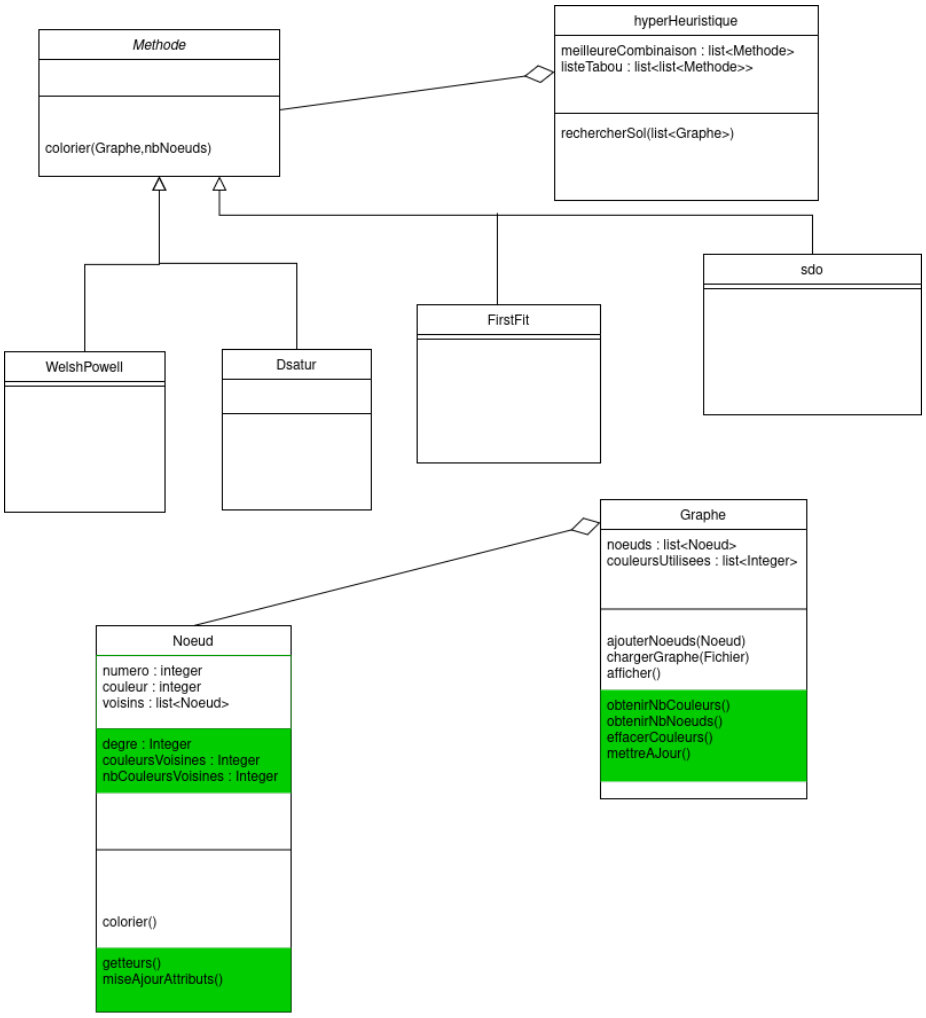

Figure 2: diagramme de classes après implémentation

6 Exemples d'éxécution

6.1 Hyper-heuristique : recherche de la meilleure combinaison sur un graphe

```
nombre de couleurs inférieur trouvé :  45 donc mise à jour de la meilleure combinaison avec :
Meilleure solution : ['FirstFit', 'sdo', 'sdo', 'WelshPowell', 'WelshPowell', 'sdo', 'Dsatur', 'WelshPowell']
nombre de couleurs inférieur trouvé :  44 donc mise à jour de la meilleure combinaison avec :
Meilleure solution : ['FirstFit', 'sdo', 'Dsatur', 'FirstFit', 'WelshPowell', 'sdo', 'Dsatur', 'WelshPowell']
nombre de couleurs inférieur trouvé :  43 donc mise à jour de la meilleure combinaison avec :
Meilleure solution : ['FirstFit', 'sdo', 'WelshPowell', 'FirstFit', 'WelshPowell', 'sdo', 'Dsatur', 'WelshPowell']
Meilleure solution : ['FirstFit', 'sdo', 'WelshPowell', 'FirstFit', 'WelshPowell', 'sdo', 'Dsatur', 'WelshPowell']
```

Figure 3: Recherche de la meilleure combinaison sur le graphe miles1000.col

6.2 Hyper-heuristique : meilleure combinaison trouvée sur chaque graphe et meilleure combinaison générale

```
Solution 0 : ['sdo', 'FirstFit', 'WelshPowell', 'WelshPowell', 'WelshPowell', 'WelshPowell', 'WelshPowell', 'WelshPowell']
Solution 1 : ['FirstFit', 'Dsatur', 'WelshPowell', 'WelshPowell', 'FirstFit', 'WelshPowell', 'sdo', 'Dsatur']
Solution 2 : ['FirstFit', 'FirstFit', 'Dsatur', 'WelshPowell', 'sdo', 'Dsatur', 'FirstFit', 'sdo']
Solution 3 : ['FirstFit', 'Dsatur', 'WelshPowell', 'Dsatur', 'Dsatur', 'FirstFit', 'Dsatur', 'Dsatur']
Solution 4 : ['sdo', 'WelshPowell', 'WelshPowell', 'WelshPowell', 'WelshPowell', 'WelshPowell', 'sdo', 'sdo']
Solution 5 : ['Dsatur', 'Dsatur', 'FirstFit', 'Dsatur', 'Dsatur', 'FirstFit', 'WelshPowell', 'FirstFit']
Solution 6 : ['WelshPowell', 'FirstFit', 'WelshPowell', 'WelshPowell', 'sdo', 'FirstFit', 'sdo', 'Dsatur']
Solution 7 : ['sdo', 'Dsatur', 'FirstFit', 'FirstFit', 'Dsatur', 'WelshPowell', 'Dsatur', 'Dsatur']
Heuristiques en position 1 : ['sdo', 'FirstFit', 'FirstFit', 'FirstFit', 'sdo', 'Dsatur', 'WelshPowell', 'sdo']
Heuristiques en position 2 : ['FirstFit', 'Dsatur', 'FirstFit', 'Dsatur', 'WelshPowell', 'Dsatur', 'FirstFit', 'Dsatur']
Heuristiques en position 3 : ['WelshPowell', 'WelshPowell', 'Dsatur', 'WelshPowell', 'WelshPowell', 'FirstFit', 'WelshPowell', 'FirstFit']
Heuristiques en position 4 : ['WelshPowell', 'WelshPowell', 'WelshPowell', 'Dsatur', 'WelshPowell', 'Dsatur', 'WelshPowell', 'FirstFit']
Heuristiques en position 5 : ['WelshPowell', 'FirstFit', 'sdo', 'Dsatur', 'WelshPowell', 'Dsatur', 'sdo', 'Dsatur']
Heuristiques en position 6 : ['WelshPowell', 'WelshPowell', 'Dsatur', 'FirstFit', 'WelshPowell', 'FirstFit', 'FirstFit', 'WelshPowell']
Heuristiques en position 7 : ['WelshPowell', 'sdo', 'FirstFit', 'Dsatur', 'sdo', 'WelshPowell', 'sdo', 'Dsatur']
Heuristiques en position 8 : ['WelshPowell', 'Dsatur', 'sdo', 'Dsatur', 'sdo', 'FirstFit', 'Dsatur', 'Dsatur']
Meilleure séquence finale : ['sdo', 'Dsatur', 'WelshPowell', 'WelshPowell', 'Dsatur', 'WelshPowell', 'sdo', 'Dsatur']
```

Figure 4: Hyper-heuristique : meilleure combinaison trouvée sur chaque graphe et meilleure combinaison générale

6.3 Affichage des résultats des coloriages des graphes tests

```
coloriage du graphe queen11_11.col avec la methode Hyper-heuristique
nbNoeuds:  121
Liste des couleurs utilisées : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
Nombre de couleurs utilisées pour le coloriage: 17
coloriage du graphe queen11_11.col avec la methode sdo
nbNoeuds:  121
Liste des couleurs utilisées : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
Nombre de couleurs utilisées pour le coloriage: 20
coloriage du graphe queen11_11.col avec la methode firstFit
nbNoeuds:  121
Liste des couleurs utilisées : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
Nombre de couleurs utilisées pour le coloriage: 17
coloriage du graphe queen11_11.col avec la methode dsatur
nbNoeuds:  121
Liste des couleurs utilisées : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]
Nombre de couleurs utilisées pour le coloriage: 17
coloriage du graphe queen11_11.col avec la methode WelshPowell
nbNoeuds:  121
Liste des couleurs utilisées : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Nombre de couleurs utilisées pour le coloriage: 19
```

Figure 5: Affichage des résultats du graphes queen11-11 avec chacunes des 5 méthodes implémentées (sdo, dsatur, WelshPowell, FirstFit, hyper-heuristique par recherche Tabou)

6.4 Exemple d'un graphe colorié obtenu

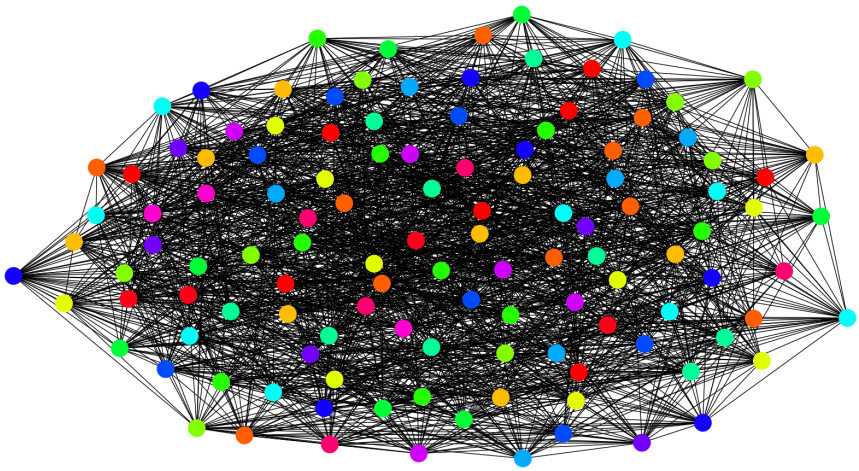

Figure 6: Graphe queen11-11 colorié par l'hyper-heuristique

7 Utilisation de ChatGPT dans ce projet

ChatGpt m'a écrit le code pour charger des instances de graphes depuis des fichiers .col, j'ai copié collé ma classe Graphe ainsi qu'un bout du fichier .col à chatGPT pour qu'il le fasse de manière adaptée à ma classe Graphe. Je lui ai aussi demandé de m'écrire un code pour afficher les graphes sous forme de dessin mais j'ai dû passer pas mal de temps à le modifier et à l'adapter à mes classes pour qu'il fonctionne. Parfois, lorsque mon code avait des erreurs je lui ai demandé d'où venait le problème si je ne le voyais pas rapidement toute seule. Il m'a aussi expliquer comment utiliser la classe Counter pour trouver les heuristiques le splus fréquentes à chaque position.

8 Tableau des Résultats

$instance - graphe$	nbS	nbA	$SolOp$	HyH	sdo	FF	$DSAT$	WP
$1.FullIns.5.col$	282	3247	8	9	8	14	14	6
$jean.col$	80	254	10	11	11	10	10	10
$mulsol.i.2.col$	188	3885	31	31	31	31	33	31
$queen11.11.col$	121	3960	11	16	20	17	17	19
$r125.1.col$	125	209	5	5	7	5	5	5
$zeroin.i.3.col$	206	3540	30	30	30	31	31	30

9 Conclusion

L'hyper-heuristique implémentée donne de bons résultats sur des graphes très différents. De plus, son temps d'éxécution est rapide car c'est une combinaison d'heuristiques simples. Néanmoins, ni les heuristiques simples, ni l'hyper heuristique ne trouve la solution optimale. La programmation générique que nous avons faite peut s'adapter à toutes sortes de problèmes pouvant se mettre sous forme d'un problème de coloration de graphes (planification et ordonnancement, attribution de fréquences en télécommunications, coloration de cartes,..) et elle peut également s'adapter à la plupart des problèmes que l'on peut mettre sous forme de graphes, pour cela il faut seulement adapter les heuristiques au problème en question. Il faut néanmoins souligner que l'utilité de l'hyper heuristique implémentée est assez limitée dans le cadre de nos exemples car les heuristiques donnaient déjà de bons résultats, mais elle permet d'obtenir de bonnes solutions pour chacunes des instances, alors que même l'heuristique qui paraît la meilleure est plus mauvaise que les autres sur certaines instances. Elle peut donc s'avérer très utile lorsque l'on a besoin de résoudre rapidement un grand nombre d'instances.

References

Aslan, M. and Baykan, N. A. (2016). A performance comparison of graph coloring algorithms. International Journal of Intelligent Systems and

Applications in Engineering, 4(Special Issue-1):1–7.

Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., and Qu, R. (2007). A graph-based hyper-heuristic for educational timetabling problems. European Journal of Operational Research, 176(1):177–192.