INF8215

TP1 - Local Search

Nom d'équipe: IA

Auteurs:

Lamia Salhi 2164386

Theo Delbecq 2161923

Professeur:

Quentin Cappart

Gaël Reynal

2020-2021

Contents

1 Introduction

L'objectif de ce TP est de créer un horaire académique sans conflits avec le moins de créneaux possible. Il y a donc un objectif de satisfaction et d'optimisation. Pour cela nous avons implémenté 3 métaheuristiques de recherche local basées sur 3 concepts différents :

- 1er choix : considérer une succession de problèmes de satisfaction à résoudre (optimisation implicite)
- 2ème choix : optimiser en ne considérant qu'un espace de solutions valides (satisfaction implicite)
- 3ème choix : considérer à la fois des solutions valides et non-valides (optimisation et satisfaction explicite)

Nous présenterons notre démarche pour chaque choix puis les résultats.

2 Présentation des modèles et démarche

2.1 Satisfaction par paliers (Choix 1)

Le premier choix de méthode de recherche se base sur le problème de décision associé, c'est à dire de satisfaire un graphe avec un nombre de couleur fixe k. On rejoint ensuite le principe d'optimisation en diminuant la valeur de k chaque fois qu'une solution faisable est trouvée pour la valeur actuelle et on renvoie la meilleure. L'espace des solutions est donc l'ensemble des assignations de couleurs possibles. La modélisation est la suivante:

- **Initialisation** : La solution initiale devant être faisable, on crée une première solution avec une méthode greedy (couleur minimale inutilisée par les voisins) ce qui définit aussi notre k initial. Pour chaque palier inférieur, on l'initialise de manière greedy aussi avec un nombre limité de k couleurs. Quand aucun choix acceptable n'est possible pour un noeud on prend celui qui minimise les conflits (avec random tie break)
- **Voisinage** : Le voisinage consiste à modifier la couleur d'un noeud. Les voisins considérés sont l'ensemble des couples noeuds/couleurs possibles dans la limite de k.
- **Évaluation** : La valeur de chaque voisin correspond au nombre de conflits (Arrêtes ayant ses deux sommets de même couleur).
- **Sélection** : On sélectionne à chaque étape le meilleur voisin, pas nécessairement améliorant.
- **Critère d'arrêt** : On arrête la recherche interne lorsque l'on trouve un voisin dont l'évaluation vaut 0 ce qui signifie que l'on a une solution satisfaite. On arrête la recherche au total lorsque le temps alloué de 20 minutes est écoulé. On renvoie alors la solution acceptable avec le k le plus faible trouvé.
- **Améliorations** : On ajoute une queue tabou de taille Random(10) + 0.6 * nb-de-conflits pour éviter les cycles (Cette taille a montré de meilleures performances qu'une taille fixe [1]). On ajoute aussi un système de mémoire pour calculer les évaluations en O(n) avec n le nombre de noeuds (plutôt que $O(n^2)$ [2].

2.2 Optimisation pure (Choix 2)

Pour le choix deux nous avons implémenté l'algorithme de hill climbing en gardant le critère de satisfaction comme contrainte dure. Le critère de satisfaction, ici, est qu'il n'y ai aucun conflits entre les cours. Voici les différentes fonctions et paramètres pour cette implémentation:

- **Initialisation** : On choisit un sous ensemble de créneaux (1/5 de l'ensemble totale dans notre cas), puis itérativement on choisit un cours aléatoirement pour lui assigner un créneau de sous-ensemble. Si il y a un conflits, on choisit une autre horaire jusqu'à que tout l'ensemble ait été balayé. Enfin, si aucun choix n'est faisable on assigne le cours à un nouveau créneau et on l'ajoute au sous ensemble.
- **Fonction de voisinage** : Nous avons considéré 2 voisinages. Le premier est celui de la chaîne de kempe qui consiste à changer le créneaux d'un cours et de tous ceux en conflits avec. Cela permet de rester dans le domaine de validité dès lors qu'on commence avec une solution faisable et, il nous permet également de minimiser la fonction d'évaluation si on sélectionne correctement le noeud de départ. Le deuxième consiste à modifier le créneaux d'un cours sélectionné parmi les créneaux présents dans le graphe. Dans notre algorithme, nous commençons par les chaînes de kempe puis une fois un minimum locale atteint on passe à ce voisinage plus large pour tenter de sortir du minima locale.
- **Fonction d'évaluation** : $f(solution) = -\sum_{i=1}^{n} |creneau_i|^2$, le nombre de créneaux totale Cette fonction permet de réduire le nombre de créneaux en défavorisant une distribution uniforme de ceux-ci.
- **Fonction de validité** : Tous les voisins qui améliorent la solution actuelle.
- **Fonction de sélection** : Pour kempe, nous sélectionnons un noeud au hasard parmi tous les noeuds ayant pour créneaux horaire celui de plus faible occurrence. Une fois le minimum local atteint, on choisi un noeud aléatoirement.
- **Critère d'arrêt** : On 'restart' notre algorithme si les résultats ne s'améliorent pas après 100 itérations après changement du voisinage. L'algorithme au complet s'arrête après 20 minutes de temps d'exécution.

Finalement, on peut voir que pour sortir d'un minima nous avons tenté de changer de voisinage et de réaliser des "restart".

2.3 Optimisation et satisfaction simultanée (Choix 3)

La dernière méthode ne fixe pas le nombre de couleurs, mais l'espace de recherche reste l'ensemble des assignations de couleurs possibles. On optimise et on satisfait le problème simultanément. La modélisation est la suivante : - **Initialisation** : On teste plusieurs méthodes d'initialisation successivement. Tout d'abord une initialisation greedy. Ensuite une initialisation totalement aléatoire en utilisant autant de couleurs que la solution greedy. Enfin, on implémente des algorithmes de colorations plus avancés tenant compte du degré des noeuds pour prioriser la coloration des noeuds centraux avec DSatur puis RLF (Recursive Largest First [3]). On initialise à une solution faisable pour au moins en avoir une à la fin.

- **Voisinage** : Le voisinage consiste à modifier la couleur d'un noeud. Les voisins considérés sont l'ensemble des couples noeuds/couleurs possibles dans la limite de k+1 avec k le nombre couleur actuel de la solution.
- **Evaluation** : La fonction d'évaluation est la suivante : $f(s) = \sum_{i=1}^{k} 2|E_i||C_i| - \sum_{i=1}^{k} |C_i|^2$ avec $|C_i|$ le nombre de noeuds de couleur i et $|E_i|$ le nombre d'arêtes dont les deux sommets sont de couleur i. Cette fonction a pour propriété qu'un minimal local sera nécessairement faisable ce qui permet de garantir qu'on se dirige vers une solution acceptable malgré la nature de la recherche.
- **Sélection** : Le voisinage étant assez grand sans possibilité de vraiment beaucoup optimiser le calcul de l'évaluation, on choisit le premier voisin améliorant. Dans le pire des cas on explore quand même tous le voisinage et si aucun voisin n'est trouvé, on est dans un minima local et la solution est forcément faisable.
- **ILS** : On ajoute aussi une mécanique d'ILS (Iterated Local Search [4]) qui permet d'augmenter la diversification pour compenser le voisinage et l'initialisation plutôt portés sur l'intensification. Une fois une solution minimale obtenue dans une recherche, on choisit 10% des couleurs au hasard que l'on retire. On reconstruit ensuite la solution. On a testé une reconstruction aléatoire et une reconstruction greedy. On relance ensuite une recherche sur la nouvelle solution. Deux critères d'acceptation, qui est le point de départ d'une boucle ILS, sont possibles : le critère de diversification qui consiste à perturber la dernière solution obtenue, et le critère d'intensification qui consiste à perturber et à repartir de la meilleure solution à chaque boucle ILS.
- **Critère d'arrêt** : Si on se trouve dans un minima local on arrête la recherche interne et on relance une boucle ILS. On mémorise la solution faisable si elle utilise moins de couleurs que la meilleure connue. On stoppe la recherche au bout de 20 minutes pour retourner la meilleure solution rencontrée.
- **Améliorations** : On ajoute une queue tabou de taille $Random(10) + 0.6 * nb - de - conflits$. La sélection rend en théorie l'utilisation d'une liste tabou peu intéressante car on ne peut pas cycler et on explore tous le voisinage dans le pire des cas cependant en pratique elle a permis d'atteindre des résultats plus rapidement dans la recherche en gagnant du temps dans l'exploration du voisinage. On ajoute aussi un système de mémoire pour calculer les évaluations en O(n) avec n le nombre de noeuds.

3 Résultats et discussions

3.1 Résultats

Résultats des métaheurisitiques par requête					
	requete A	requete B	requete C	requete D	requete E
choix 1	4	5	18	32	33
choix 2	4	5	18	31	54
choix 3	4	5	15	31	31

3.2 Discussion

On obtient l'optimal pour A et B assez rapidement pour chaque méthode. Une initialisation greedy permet même, d'obtenir ces résultats directement et la recherche n'a finalement que peu d'intérêt.
Pour le choix 1, on obtient 22 pour l'instance C avec la première version qui est la valeur d'initialisation. L'ajout d'optimisation de calcul de l'évaluation du voisinage permet de multiplier par 6 le nombre de passages de boucles (40 pour l'instance E). On arrive à 19 couleurs. La liste tabou permet d'atteindre 18 en limitant les cycles. Pour les instances D et E on reste valeurs d'initialisation ce qui semble montrer que la recherche n'arrive pas à sortir d'un minima local. Le voisinage est grand et ne permet que de petits déplacements. Une solution complexe n'est donc pas forcément facilement atteignable en 20 minutes.
Pour le choix 2, l'initialisation nous permet de commencer à 33 créneaux pour en finir avec 18 à la requête C, tandis que pour le requête D on commence à 42 créneaux pour arriver à 31 et enfin pour la requêtes E on commence à 128 pour arriver à 54. Ces résultats restent corrects car il dépassent les résultats du greedy pour la C et la D. Cependant les résultats pour la E sont médiocres.
On obtient de meilleur résultat avec la dernière version du choix 3. La solution totalement aléatoire (init et reconstruction aléatoire, diversification acceptance) est la moins bonne. Elle permet d'obtenir 19 couleurs avec C et n'améliore pas l'initialisation sur D et E. La solution greedy avec critère d'intensification permet de passer à 18 sur C et à 31 et 32 sur D et E respectivement. Le critère de diversification pour cette solution permet d'atteindre 17 pour C mais on reste à 33 pour D et E. Cela montre à nouveau que l'initialisation met la recherche dans un minima difficile à fuir. La diversification aide pour l'instance C mais étant donné que l'on fait beaucoup moins d'itérations pour les instances D et E de par leur complexité, la diversification n'a pas le temps de porter ses fruits face à l'intensification. Finalement, les performances sont très dépendantes de l'initialisation. On arrive avec 15 au mieux (parfois 16 ou 17) pour C et 31 pour D et E avec une initilisation par DSatur puis RLF.

4 Conclusion

La solution retenue est donc le choix trois avec une configuration greedy (Initialisation RLF, Tabu queue, greedy perturbation, intensification acceptance).
Finalement, ce TP nous aura permis de réaliser que le principal challenge des algorithmes de métaheuristique est de sortir des minima locaux et de l'importance d'une bonne initialisation. Nous avons pu construire 3 algorithmes basés sur des concepts différents.

5 Références

[1] Chiar, Marco Dumitrescu, Irina Stützle, Thomas. (2003). Local Search for the Colouring Graph Problem. A Computational Study.

[2] Fleurent, C., Ferland, J.A. Genetic and hybrid algorithms for graph coloring. Ann Oper Res 63, 437–461 (1996).

[3] Adegbindin, Mourchid Hertz, Alain Bellaiche, Martine. (2016). A new efficient RLF-like algorithm for the vertex coloring problem. Yugoslav Journal of Operations Research.

[4] Chiar, Marco Stützle, Thomas Intellektik, Fachgebiet. (2003). An application of Iterated Local Search to Graph Coloring Problem.