Analyse d'algorithmes - Travail Pratique (Première partie)

Étude d'un problème NP-complet : 3-Coloration

Organisation

- Ce TP est à réaliser seul ou en binôme.
- Deux séances de 3h30 y sont dédiées.

Attention : de nouveaux objectifs seront ajoutés au début de la deuxième séance de TP.

Problème 3-COLORATION:

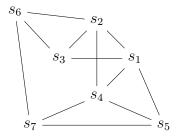
Dans le cadre de la théorie des graphes, un graphe est 3-coloriable si on peut colorier tous ses sommets avec seulement trois couleurs de façon à ce que deux sommets voisins ne soient jamais de la même couleur. Le graphe



est 3-coloriable:



Alors que le graphe



n'est pas 3-coloriable (essayer un peu de le 3-colorer pour vous en convaincre). Le problème 3-coloration (nommé 3C) est : donné un graphe en entrée, peutil être colorié avec seulement trois couleurs?

Objectif

Réaliser les quatre programmes ci-dessous (plus de détails sur ces programmes dans la suite) :

• Verificateur: On a vu en cours qu'un problème est dans NP s'il existe un algorithme polynomial pouvant vérifier qu'une solution du problème est bien une solution. Ici, la solution sera juste une "coloration du graphe".

Entrée Un graphe et une coloration de chaque sommet du graphe.

Sortie Accepter si la coloration est une 3-coloration valide

• SolvBackTracking : Un algorithme intuitif pour résoudre le problème.

Entrée Un graphe donné par un fichier externe.

Sortie Le programme résout le problème 3C via backtracking.

• SolvRandom: On se propose d'implémenter l'algorithme probabiliste suivant: commencer avec une coloration aléatoire des sommets. Puis, faire n fois: tant que ce n'est pas une coloration valide, choisir aléatoirement un des sommets problématiques et le changer de couleur. Comparer cet algorithme avec l'algorithme précédent.

Entrée Un graphe donné par un fichier externe.

Sortie Le programme résout le problème 3C en utilisant l'heuristique ci-dessus.

• SolvLawler: On se propose d'implémenter une version simple de l'algorithme de Lawler. Pour tout sous-ensemble S de sommets, on teste si S est un ensemble indépendant et si $G \setminus S$ est un ensemble biparti. Quelle est la complexité de cet algorithme ?

Entrée Un graphe donné par un fichier externe.

Sortie Le programme résout le problème 3C en utilisant l'algorithme proposé.

Détails sur les programmes

Les programmes que vous écrivez doivent respecter les contraintes suivantes :

• Vous pourrez définir des objets "Graphe" et "Coloration". Le premier algorithme prendra en entrée un graphe et une coloration et testera s'il s'agit d'une 3-coloration valide. Cette fonction doit posséder une complexité temporelle polynomiale en fonction de la taille de l'instance.

- Mais, les trois derniers programmes (SolvBackTracking, SolvRandom et SolvLawler) doivent lire une instance du problème sur un fichier externe ou sur l'entrée standard (pensez à réutiliser votre code). L'entrée sera donnée par un fichier texte encodant une instance de graphe. Des exemples sont sur le wiki. En outre, un générateur de telles entrées est aussi fourni.
- Description plus détaillée des trois derniers algorithmes :
 - 1. Le principe du backtracking est de construire graduellement une solution en faisait des choix plus ou moins arbitraires et en changeant ces choix lorsqu'on ne peut plus progresser ("backtrack"). Plus précisément, on essaie de créer une solution de 3C (x1, x2,...,xn). Un algorithme de type backtracking va attribuer une valeur (i.e., une couleur) à x1, puis une valeur à x2, puis à x3 et ainsi de suite. À chaque étape on teste si la solution partielle (x1, x2,...,xi) contient une contradiction. Si c'est le cas, on change la valeur de xi et on teste à nouveau. Si toutes les valeurs possibles de xi ont été testées et qu'aucune n'est possible, alors on remonte à xi-1 et lui attribue une autre valeur.

L'algorithme termine lorsque x_n s'est vu attribué une valeur et que toutes les valeurs (x_1, \ldots, x_n) sont cohérentes. Si toutes les valeurs ont été testées et qu'aucune ne produit une solution, on peut donc affirmer qu'il n'existe pas de solution.

2. L'algorithme suivant est un algorithme probabiliste qui ne va pas toujours être correct mais qui est ce coup-ci rapide. La qualité d'un tel algorithme sera surtout mesurée en fonction de son taux de réponses correctes.

On propose ici une version basique d'un tel algorithme :

Générer une première coloration aléatoire (avec potentiellement des sommets voisins de la même couleur)

Trouver les arêtes problématiques et les stocker dans une variable ${\cal K}$

Répéter un certain nombre de fois l'étape de correction : choisir aléatoirement une arête problématique et colorier différemment un de ses extrémités (extrémité et couleur choisies aléatoirement). On met à jour K.

Si à un moment |K| = 0, on s'arrête : on a trouvé une coloration.

De nombreuses améliorations peuvent être apportées [mais non demandées]:

- L'algorithme précédent risque fort de tomber dans des minimums locaux, pour éviter cela, on peut répéter l'algorithme entier un certain nombre de fois.
- Au lieu de partir d'une coloration initiale complètement aléatoire, on pourrait commencer par une tentative de bonne coloration.
 On choisit un sommet de grand degré et on essaye de colorer les autres sommets de proche en proche de manière la plus valide possible.
- Dans la boucle, au lieu de choisir directement un sommet et une couleur aléatoirement, on commence par chercher s'il n'y a pas un choix de sommet et de couleur qui diminue |K|.
- On peut trouver des améliorations plus compliquées (chercher par exemple les techniques de "Backbone, Coarsening" sur internet).
- 3. Le dernier algorithme est une simplification de l'algorithme de Lawler. Si un graphe est 3-colorié, chaque couleur correspond à un ensemble indépendant. Si on retire tous les sommets d'une couleur, le graphe devient biparti. Or tester si un graphe est biparti peut se faire efficacement.
 - En fait dans l'algorithme original, au lieu de considérer tous les sous-ensembles S de sommets, l'idée est de boucler sur les ensembles indépendants maximaux S.

Documentation

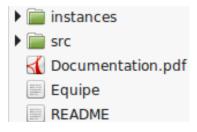
En plus du guide de compilation (le fichier README) vous devez rendre un document pdf ayant comme contenu :

- 1. Une page titre.
- 2. Une description de votre problème (avec au moins un exemple).
- 3. Une analyse de la complexité temporelle de chaque programme. Pour ce faire, découpez votre programme en ses principales parties et pour chacune d'elles, donnez une borne en notation grand-O et justifiez votre analyse.
- 4. Il serait très apprécié que vous testiez un peu la correction (est-ce qu'il se plante souvent?) de l'algorithme aléatoire.

Livrables

Rendez une archive (au format tar, tgz ou zip uniquement). Cette archive doit contenir:

• Dossier instances : ce dossier contient des exemples d'instance de votre problème. Au minimum, il doit contenir deux instances vrai et deux instances fausse. Ces instance doivent être dans des fichiers séparés et leurs noms doivent être représentatifs (ex : "3C_vrai").



- Dossier src : dossier contenant les sources de vos programmes.
- Fichier Documentation.pdf: un document décrivant vos algorithmes (voir la section Documentation ci-dessous.)
- Fichier Équipe : document texte contenant les noms et les adresses courriel de chacun des membres de l'équipe.
- Fichier README : document texte expliquant comment compiler et lancer vos trois programmes.