TP2 Analyse d’Algorithmes

Hugo Rey - Simon Pichenot

Partie 1 - 3-colorisation

Dans cette première partie on utilise différents algorithmes permettant de trouver si un graph est 3-coloriable. C'est-à-dire que chaque sommet du graph prend une couleur parmi trois et aucun de ces voisins n'a la même couleur que lui.

Backtracking

La complexité totale de la fonction **solve_backtracking_rec** dépend du nombre de nœuds à colorier restant x le nombre de fois on on recommence car les choix n’étaient pas bons. Donc x * n où x est le nombre de répétitions donc **O(n)**.

Random

La complexité globale du programme dépendra du nombre d'itérations de la boucle while. Dans le pire des cas, si le graphe est très difficile à colorier, la boucle peut s'exécuter un grand nombre de fois, ce qui rendra la complexité exponentielle. Cependant, en moyenne, la complexité sera généralement dominée par les étapes linéaires liées au nombre de sommets du graphe. Donc, une estimation globale de la complexité serait généralement en O(n) pour la plupart des cas, où n est le nombre de sommets du graphe.

Lawler

Nous avons commencé à implémenter l'algorithme mais nous avons été bloqués et n’avons pas pu finir.

Partie 2 - Sat-Solver

Problème de clique

Ce problème nous a permis de comprendre comment se servir du sat solver fourni par le professeur. Le sat solver récupère une liste à deux dimensions de variables (ie : [[1,2],[3,4]]). Chaque liste de la deuxième dimension est une expression logique ou chacune des variables est séparée par un “ou”. La première dimension est une expression logique qui fait le lien entre chacune des expressions vu précédemment en les connectant avec des “et”

Retour sur 3-colorisation

Nous avons transformé le probleme de 3-colorisation en un problème solvable par le sat solver en créant les contraintes suivantes qui doivent toutes être respectée :

- Contrainte 1 : Chaques noeud du graph doit avoir une des 3 couleur possibles
- Contrainte 2 : Chaques noeud doit avoir une couleur différente de ses voisins

Dans l'exemple suivant : [[1,2,3],[4,5,6]] (contrainte 1)
1 : Noeud 1 est rouge
2 : Noeud 1 est bleu
3 : Noeud 1 est vert
4 : Noeud 2 est rouge
5 : Noeud 2 est bleu
6 : Noeud 2 est vert

donc (1 ou 2 ou 3) ET (4 ou 5 ou 6)

Couverture par 3 cliques

On peut dire qu'un problème est np-complet en utilisant un autre problème np-complet. Pour cela il faut prouver qu'un problème B est au moins aussi difficile qu'un problème np-complet A. Pour ce faire, on transforme un instance du problème B en une instance du problème A en un temps polynomial.
Dans notre cas nous avons transformé une instance du problème 3 cliques en une instance du problème 3-colorisation de manière à résoudre notre problème. Donc si 3-colorisation est np-complet 3 cliques l'est aussi.