TD 05 – Modélisation, approximation

Exercice 1. *Modélisation 3-Coloration*

Le problème **3-Coloration** consiste, étant donné un graphe non-orienté $G = (V, E)$, à décider si on peut assigner à chacun de ses sommets une couleur parmi $\{1, 2, 3\}$, de façon à ce que si deux sommets sont reliés par une arête alors ils ont des couleurs différentes.
Mathématiquement, $\exists c : V \to \{1, 2, 3\}, \forall \{u, v\} \in E, c(u) \neq c(v)$?

1. Expliquer comment réduire **3-Coloration** à **SAT**, c'est-à-dire, à partir d'une instance G de **3-Coloration**, comment construire une formule φ_G telle que la réponse à **SAT** donne la réponse à **3-Coloration** (qui soit satisfaisable si et seulement si le graphe est 3-coloriable).
   (a) Quelles variables ?
   (b) Quelle conjonction de contraintes (clauses) ?
   (c) Exprimer en CNF.

   Donner le nombre de variables, ainsi le nombre et la taille des clauses, comme une fonction de $n = |V|$ et $m = |E|$.

On peut alors utiliser un solveur SAT pour résoudre **3-Coloration**.

2. Donner un algorithme prenant en entrée une instance G du problème **3-Coloration** (G encodé par matrice d'adjacence), et qui produit en sortie φ_G au format DIMACS.
   Évaluer sa complexité.

Exercice 2. *Approximation*

Source : Cormen problème 35-1.

Le problème d'optimisation du **Sac-à-dos** consiste, étant donnés n objets de tailles $e_1, \ldots, e_n \in \mathbb{N}$ et une taille de sac à dos p, à minimser le nombre de sac à dos utilisés pour ranger les n objets (un sac à dos ne pouvant contenir qu'un ensemble d'objets dont la somme des tailles est $\leq p$).
L'entrée est codées comme une liste de nombres en binaires.

1. Expliquer comment résoudre **Partition** à l'aide d'un algorithme pour **Sac-à-dos**. Analyser le temps de calcul de la transformation (réduction).

Partition est un problème de décision NP-complet, donc **Sac-à-dos** est un problème d'optimisation NP-dur (il n'est pas dans NP car ici ce n'est pas un problème de décision).

2. Décrire le fonctionnement de l'algorithme *first-fit* pour le problème du **Sac-à-dos**.

Soit $S = \sum_{i=1}^{n} e_i$.

3. Donner une borne inférieure au nombre optimal de sac à dos pour une instance donnée.
4. Expliquer pourquoi l'algorithme *first-fit* laisse au plus un sac à dos rempli à moins de la moitié de sa capacité p. Indice : par induction sur le nombre n d'objets.
5. En déduire que l'algorithme *first-fit* n'utilise jamais plus de $\left\lceil \frac{2S}{p} \right\rceil$ sacs à dos.
6. Proposer une implémentation efficace de l'algorithme *first-fit* pour le problème d'optimisation du **Sac-à-dos**, et analyser son temps de calcul.