TD 03 – Formalisation, transversal, X-SAT faciles

Exercice 1. *Formalisation de problèmes NP ou coNP*

Voici des problèmes exprimés en français, à formaliser à l'aide de graphes.
Pour chacun, dire s'il appartient à la classe NP ou à la classe coNP (ou les deux).

1. Une ville est composée d'îlots reliés par des ponts. On souhaite décider s'il existe une balade passant exactement une fois par chaque pont avant de revenir à son point de départ.
2. Une ville est composée d'îlots reliés par des ponts. On souhaite décider s'il existe une balade passant exactement une fois par chaque îlot avant de revenir à son point de départ.
3. On a une liste de cours de 4 heures (une demi-journée), chacun donné par un enseignant, et suivi par une liste d'étudiants. Un enseignant ne peut pas donner deux cours simultanément, ni un étudiant suivre deux cours simultanément. Le problème consiste à décider si un emploi du temps sur k demi-journées peut être établi.
4. Un interblocage intervient lorsque des processus concurrents s'attendent mutuellement (par exemple pour utiliser des ressources détenues par d'autres). Étant donnée une telle situation, on souhaite décider si elle peut être guérie en tuant au plus k processus.
5. Des vendeurs proposent un puzzle composé de n^2 pièces carrées dont les bords sont colorés. Le jeu consiste à remplir un plateau de taille $n \times n$, de façon à ce que deux pièces adjacentes aient la même couleur sur le côté qu'elles partagent. Un prix de 1 000 000 $ sera offert à la première personne qui réussira ce puzzle. On souhaite décider si les vendeurs de ce jeu sont malhonnêtes ? On demande seulement si c'est dans NP ou coNP.

Exercice 2. *Transversal (*vertex cover*)*

Dans un graphe non-orienté $G = (V, E)$, un transversal est un sous-ensemble $X \subseteq V$ tel que pour toute arête $\{u, v\} \in E$, on ait $u \in X$ ou $v \in X$. Autrement dit, un transversal est un sous-ensemble de sommets partageant au moins un sommet avec chaque arête du graphe.

1. Donner un algorithme prenant en entrée un graphe G à n sommets encodé par sa matrice d'adjacence, et un sous-ensemble de sommets X encodé par un tableau de n booléens, et qui vérifie si X est un transversal de G. Évaluer sa complexité.
2. Donner un algorithme qui, sur la même entrée, vérifie si X est un transversal minimal, c'est-à-dire qu'il ne contient aucun transversal strictement plus petit (aucun sous-ensemble strict de X n'est un transversal). Évaluer sa complexité.
3. Donner un algorithme prenant en entrée uniquement un graphe G, et qui retourne un transversal minimal de G. Évaluer sa complexité.
4. Donner un algorithme prenant en entrée un graphe G et un entier k, et qui teste si G possède un transversal de taille k ou moins. Évaluer sa complexité.
5. Supposer que nous ayons accès à une implémentation `stable(`G`,`p`)` résolvant efficacement le problème de décider si G possède un ensemble de sommets indépendants (sans aucun lien entre eux) de taille p ou plus. Utiliser un appel à `stable` pour résoudre le problème précédent de décider si G possède un transversal de taille k ou moins.

Exercice 3. *X-SAT faciles*

On s'intéresse à des variantes de SAT résolues en temps polynomial.
Une clause de Horn est une clause possédant au plus 1 littéral positif. Le problème **Horn-SAT** consiste à décider la satisfiabilité d'une formule avec uniquement des clauses de Horn.

1. Que dire de la satisfiabilité d'une instance de Horn-SAT, si aucune clause n'est d'arité 1 ?
2. En déduire un algorithme de résolution du problème Horn-SAT en temps poly.

Une instance de **2-SAT** n'a que des clauses de taille 2. Afin de satisfaire la formule, pour chaque clause $(\ell_i \vee \ell_j)$, si ℓ_i ne satisfait pas la clause, alors ℓ_j doit satisfaire la clause.

3. Construire un graphe orienté G_φ avec 2 sommets pour chaque variable, représentant ces déductions « si ... alors ... » pour toutes les clauses d'une instance φ du problème 2-SAT.
4. Comment utiliser le graphe G_φ pour décider la satisfiabilité de φ ?

Exercice 4. *Attention*

✎ Montrer que l'algorithme suivant est de complexité exponentielle.

```
Entrée : un entier x en binaire.
y = log2(x)
pour i de 1 à y faire:
    x = x*x
retourner x
```