# COMPLEX – Complexité, algorithmes randomisés et approchés
# Enoncés de TD (1 à 5)

**Année 2018–2019**


***Équipe enseignante :***

*Bruno Escoffier*
*Fanny Pascual*
*Alexandre Teiller*

# Table des matières

# 1 Machines de Turing et complexité de problèmes

## Exercice 1 Complexité d'algorithmes - Rappels

**Q 1.1** Evaluer la complexité d'un algorithme qui à partir de deux listes triées A et B construit une liste unique contenant les éléments des deux listes A et B.

**Q 1.2** Etant donné $n$ points dans un plan, évaluer la complexité d'un algorithme qui calcule la paire de points les plus proches.

**Q 1.3** Etant donné un tableau trié d'entiers, évaluer la complexité d'un algorithme qui teste si l'entier $x$ est contenu dans le tableau.

**Q 1.4** Etant donné $n$ sous-ensembles de $\{1, 2, \cdots, n\}$, évaluer la complexité d'un algorithme qui teste l'existence d'une paire de sous-ensembles disjoints.

**Q 1.5** Etant donné un graphe $G$ et une constante $k$, évaluer la complexité d'un algorithme qui teste l'existence de $k$ sommets deux à deux non adjacents (non reliés par une arête) dans $G$.

**Q 1.6** Etant donné un graphe $G$, évaluer la complexité d'un algorithme qui retourne un sous-ensemble de cardinalité maximale de sommets deux à deux non adjacents dans $G$.

**Q 1.7** Etant donné un ensemble $S$ de $n$ entiers et un entier $x$, évaluer la complexité d'un algorithme qui détermine s'il existe deux éléments de $S$ dont la somme vaut exactement $x$.

## Exercice 2 Tiré de l'examen réparti 1 2014-2015

**Q 2.1** On considère la machine de Turing suivante, où $q_0$ est l'état initial, $q_a$ l'état d'acceptation et $q_r$ l'état de rejet. Quel est le langage reconnu (l'alphabet étant $\{a, b\}$ pour les mots) ?

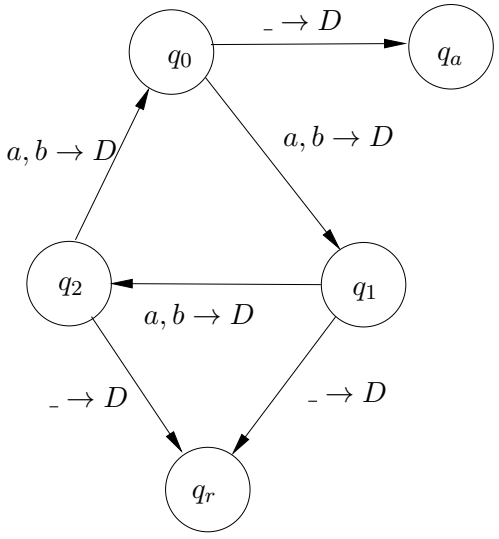



FIGURE 1 – Machine de Turing

## Exercice 3 Machines de Turing

Pour chacun des problèmes suivants, donner une description haut-niveau d'une machine de Turing le résolvant. Pour le problème de votre choix parmi ces problèmes, décrire formellement une machine de Turing le résolvant.

**Q 3.1** Etant donné un mot $w$ constitué de 0 et de 1, la question est de savoir si $w$ contient au moins un 0.

**Q 3.2** Etant donné un mot $w$ constitué de 0 et de 1, la question est de savoir si $w$ contient autant de 0 que de 1.

**Q 3.3** On considère le problème suivant : étant donné un mot $w$ constitué de 0 et de 1, la question est de savoir si $w$ est un palindrome ou non.
*Rappel :* un palindrome est un mot dans lequel l'ordre des lettres reste le même qu'on le lise de gauche à droite ou de droite à gauche (par exemple *kayak* et*010010* sont des palindromes).

**Q 3.4** Etant donné un mot $w$ constitué de 0, la question est de savoir si la longueur de $w$ est une puissance de 2 (i.e. la machine de Turing reconnaît le langage $\{0^{2^n} | n \geq 0\}$).

## Exercice 4 Tiré de l'examen réparti 1 en 2015-2016

On s'intéresse au problème suivant : étant donné un mot $w$ écrit sur l'alphabet $\{0, 1\}$, la question est de savoir si $w$ est constitué d'un certain nombre de 0 suivis d'un nombre identique de 1. Autrement dit, ce problème consiste à reconnaître les mots du langage $L = \{0^n 1^n | n \geq 0\}$.

**Q 4.1** Donner une description haut-niveau d'une machine de Turing résolvant ce problème.

**Q 4.2** Décrire formellement (en donnant le diagramme d'états notamment) une machine de Turing résolvant ce problème.

## Exercice 5 Le problème de l'arrêt

Une machine de Turing prend en entrée une chaîne (finie) de caractères $x = x_0 x_1 \dots x_{n-1}$. A l'exécution, trois comportements sont possibles :

- La machine s'arrête dans l'état d'acceptation ;
- La machine s'arrête dans un autre état ;
- La machine de s'arrête pas (boucle infinie).

Notons qu'une machine de Turing *ALGO* est également une chaîne de caractères, qui représente le codage de *ALGO*, et qui sera notée *algo*.

On considère le problème suivant, consistant à déterminer si une machine de Turing s'arrête sur un mot donné ou pas. Plus précisément :

- L'entrée est constituée d'un couple $(algo, x)$, où *algo* est un mot codant une machine de Turing *ALGO*, et $x$ un mot.
- L'entrée doit être acceptée si et seulement si *ALGO* s'arrête sur $x$.

Supposons qu'il existe une machine de Turing *ARRET* ($\rightarrow$ un algorithme) qui décide ce problème.

Considérons maintenant la machine de Turing suivante, appelée $PARADOXE$, et qui prend en entrée le codage *algo* d'une machine de Turing $ALGO$. La machine $PARADOXE$ est définie de la façon suivante :

1. Si $ARRET(algo, algo)$ est accepté, alors boucler indéfiniment ;
2. Sinon accepter.

**Q 5.1** Appliquer $PARADOXE$ sur l'entrée *paradoxe*. Que se passe-t-il ? Qu'en déduisez-vous ?

---

**Exercice 6 2-Sat**

---

Un *littéral* est une variable booléenne, ou la négation d'une variable booléeenne (par exemple $x$ ou $\overline{x}$). Une *clause* est composée de plusieurs littéraux liés par des $\vee$ (par exemple $x_1 \vee \overline{x_2} \vee x_3$). Une formule booléenne est en forme normale conjonctive (FNC) si elle est constituée de plusieurs clauses liées par des $\wedge$ (par exemple $(x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_3})$). Une formule booléenne 2FNC est une formule booléenne en forme normale conjonctive telle que chaque clause a 2 littéraux. Par exemple $(x_1 \vee \overline{x_2}) \wedge (x_1 \vee x_3) \wedge (\overline{x_1} \vee x_2) \wedge (x_2 \vee \overline{x_4})$ est une formule booléenne 2FNC.
Le problème 2-SAT est le suivant :

**Entrée :** une formule booléenne 2FNC $\phi$
**Question :** $\phi$ est-elle satisfiable ?

**Q 6.1** Soit $n$ le nombre de variables de $\phi$ (on suppose sans perte de généralité que ces variables sont $x_1, \ldots, x_n$), et $m$ le nombre de clauses. Quelle est la complexité d'un algorithme qui teste chaque valeur de vérité une par une pour déterminer si $\phi$ est satisfiable ?

**Q 6.2** On cherche maintenant à résoudre le problème avec un algorithme de complexité polynomiale. On construit le graphe orienté $G(\phi)$ de la façon suivante :

— $G$ a $2n$ sommets : un sommet pour chaque variable $(x_i)$ et un sommet pour la négation de chaque variable $(\overline{x_i})$.
— Pour chaque clause $(a \vee b)$ de $\phi$, où $a$ et $b$ sont des littéraux, on crée l'arc $(\overline{a}, b)$ et l'arc $(\overline{b}, a)$. Ces arcs signifient que si $a$ est faux alors $b$ doit être vrai, et réciproquement.

Soient $\phi_1 = (x_1 \vee x_2) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee x_1)$ et $\phi_2 = (\overline{x_1} \vee x_2) \wedge (x_1 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (x_1 \vee x_3)$. Représenter les graphes $G(\phi_1)$ et $G(\phi_2)$.

**Q 6.3** Montrer qu'une formule $\phi$ est satisfiable si et seulement si il n'existe pas de variable $x_i$ telle que :

1. il existe un chemin entre $x_i$ et $\overline{x_i}$ dans $G(\phi)$, et
2. il existe un chemin entre $\overline{x_i}$ et $x_i$ dans $G(\phi)$.

**Q 6.4** En déduire que 2-SAT$\in$ P.

**Q 6.5** Est-il possible d'utiliser la même technique pour résoudre en temps polynomial le problème 3-SAT ?
*Rappel :* Problème 3-SAT
**Entrée :** une formule booléenne 3FNC $\phi$ (une formule booléenne 3FNC est une formule booléenne en forme normale conjonctive telle que chaque clause a 3 littéraux)
**Question :** $\phi$ est-elle satisfiable ?

## Exercice 7 Ordonnancement de tâches

On souhaite exécuter sur une ou plusieurs machine(s) un ensemble $S = \{J_1, J_2, \ldots, J_n\}$ de $n$ tâches de durées respectives $p_1, p_2, \cdots, p_n$. Deux tâches ne peuvent être exécutées en même temps sur la même machine. La date de fin $C_i$ de la tâche $J_i$ correspond à la date à laquelle la tâche se termine. On considère que la première tâche commence à être exécutée à la date 0. Un *ordonnancement* est une affectation de chaque tâche à une machine et à une date de début d'exécution respectant les contraintes ci-dessus (à chaque instant au plus une tâche est exécutée sur chaque machine, la première tâche commence à être exécutée à la date 0).

**Q 7.1** On considère que l'on dispose d'une seule machine. Quel est l'ordre optimal d'exécution des tâches si l'on souhaite minimiser la somme des dates de fin des tâches (ou durée moyenne de réalisation) $\sum_{i=1}^{n} C_i$ ? Quelle est la complexité d'un algorithme ordonnançant les tâches de façon à minimiser la somme des dates de fin des tâches ?

**Q 7.2** On considère toujours que l'on dispose d'une seule machine, mais on suppose maintenant qu'un poids positif (ou priorité) $w_i$ est associé à chaque tâche $J_i$, et que l'on souhaite minimiser la somme pondérée des dates de fin des tâches $\sum_{i=1}^{n} w_i C_i$. Montrer que l'ordre optimal est obtenu en séquençant les tâches dans l'ordre croissant des $\frac{p_i}{w_i}$ (ordre WSPT, pour "Weighted Shortest Processing Times").

**Q 7.3** En déduire une solution optimale et son coût pour l'exemple donné dans le tableau ci-dessous.

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $p_i$ | 8 | 6 | 3 | 7 | 4 | 8 |
| $w_i$ | 8 | 3 | 6 | 7 | 8 | 1 |

Combien existe-t-il de solutions optimales pour cet exemple ?

**Q 7.4** On considère maintenant que l'on dispose de plusieurs machines parallèles. Sur plusieurs machines, l'algorithme ordonnançant les tâches est le suivant : étant donné un ordre fixé des tâches (une liste de priorité), dès qu'une machine est disponible (i.e. n'exécute aucune tâche), alors on lui affecte la première tâche non encore ordonnancée dans la liste de priorité.
Les règles de priorité énoncées dans les questions 1 et 2 sont-elles toujours valables ?

## Exercice 8 Primalité

Le problème Premier consiste à déterminer, étant donné un entier $n \geq 2$, si $n$ est premier ou pas. Le problème Composé consiste à determiner si $n$ est composé (si $n$ peut s'écrire $n = pq$ avec $p$ et $q$ deux entiers supérieurs ou égaux à 2).

**Q 8.1** On propose l'algorithme suivant :

$p \leftarrow true$
**Pour** $i$ allant de 2 à $n-1$ **faire**
**Si** $n \mod i = 0$ **alors** $prem \leftarrow faux$
**Fin Pour**

Cet algorithme permet-il d'affirmer que Premier est dans P ?

**Q 8.2** Donner un certificat montrant que Composé est dans NP. Que pouvons-nous en déduire pour Premier ?

**Q 8.3** Pouvez-vous donner un certificat montrant que Premier est dans NP ?

## Exercice 9 Sac à dos

Le problème SAC-À-DOS est le suivant :
**Entrée :**
- Un ensemble de $n$ objets $S = \{1, \ldots, n\}$, chaque objet $i$ ayant un poids $w_i \in \mathbb{N}$ et rapportant un profit $p_i \in \mathbb{N}$.
- Un sac à dos supportant un poids $W \in \mathbb{N}$.

**Question :** trouver un sous-ensemble $S'$ des objets de $S$ tel que les objets rentrent dans le sac-à-dos ($\sum_{i \in S'} w_i \leq W$) et le profit engendré par ces objets ($\sum_{i \in S'} p_i$) est maximisé.

**Q 9.1** Votre voisin(e) propose d'utiliser l'algorithme suivant : il suffit de trier les objets par rapport $\frac{p_i}{w_i}$ décroissant, puis de prendre ensuite les objets de manière gloutonne dans cet ordre. Montrer que cet algorithme ne retourne pas toujours la solution optimale.

**Q 9.2** Donner le problème de décision associé à ce problème, et montrer qu'il est NP-complet. Vous pourrez supposer que le problème PARTITION est NP-complet.

*Rappel :* le problème PARTITION est le suivant :
**Entrée :** un ensemble de $n$ entiers positifs $S = \{x_1, x_2, \ldots, x_n\}$.
**Question :** existe-t-il un sous-ensemble $P$ de $S$ tel que $\sum_{x_i \in P} x_i = \sum_{x_i \in S \setminus P} x_i$ ?

## Exercice 10 3-SAT

On considère le problème SAT, ainsi que le problème ATMOST3-SAT, restriction de SAT aux instances où toutes les clauses sont de taille au plus 3 (contiennent au plus 3 littéraux).
On souhaite montrer que ATMOST3-SAT est NP-complet, par réduction depuis SAT.

**Q 10.1** Considérons $c = (\ell_1 \vee \ell_2 \vee \ell_3 \vee \ell_4)$ une clause de taille 4 d'une instance de SAT sur les variables $\{x_1, \cdots, x_n\}$ ($\ell_i$ est un littéral : soit une variable $x_j$ soit sa négation $\overline{x_j}$).
Soit $z_1, z_2, z_3$ trois nouvelles variables : on considère les 4 clauses $c_1 = (\ell_1 \vee z_1)$, $c_2 = (\overline{z_1} \vee \ell_2 \vee z_2)$, $c_3 = (\overline{z_2} \vee \ell_3 \vee z_3)$, $c_4 = (\overline{z_3} \vee \ell_4)$.
Etant donnée une valeur de vérité sur les variables $x_j$, montrer que cette valeur vérifie $c$ si et seulement s'il est possible de donner une valeur de vérité aux variables $z_i$ de manière à ce que les 4 clauses $c_1, c_2, c_3, c_4$ soient vérifiées.

**Q 10.2** Généraliser le résultat précédent à une clause de taille $k \geq 4$.

**Q 10.3** En déduire que ATMOST3-SAT est NP-complet.

## Exercice 11 Stable (1)

Le problème du STABLE est le suivant :
**Entrée :** Un graphe $G = (V, E)$ et un entier $k$.
**Question :** Existe-t-il un ensemble de sommets $V' \subset V$ , de taille $k$ tel que deux sommets de $V'$ ne sont jamais reliés entre eux : $V' \times V' \cap E = \emptyset$ ?

On souhaite montrer que ce problème est NP-complet, par réduction de 3-SAT.

**Q 11.1** Montrer que le problème du STABLE appartient à NP.

**Q 11.2** On se donne une formule $\phi$ de 3-SAT de la forme $\phi = C_1 \wedge C_2 \wedge \ldots C_m$ avec $C_i = y_{i1} \vee y_{i2} \vee y_{i3}$.

On construit le graphe $G(\phi)$ dont les sommets sont les littéraux $y_{ij}$, et les arêtes sont d'une part les couples $(y_{ij}, y_{ik})$ pour $j \neq k$ et d'autre part les couples $(y_{ij}, y_{lp})$ tels que $i \neq l$ et $y_{ij} = \overline{y_{lp}}$. Montrer que $G(\phi)$ contient un Stable de taille $m$ si et seulement si $\phi$ est satisfiable.

**Q 11.3** En déduire que le problème du STABLE est NP-complet.

## Exercice 12 Stable (2)

On suppose maintenant que l'on sait que le problème de la CLIQUE est NP-complet.

**Q 12.1** Montrer que le problème du STABLE est NP-complet, par réduction de CLIQUE.

## Exercice 13 Le problème du mètre pliant de charpentier

Un charpentier a acheté un mètre pliant ayant des sections de longueurs variables. Ce mètre, possédant $s$ sections, peut être représenté par $s$ nombres $a_1, \ldots, a_s$, la longueur de la $i^{\text{ème}}$ section étant $a_i$ (les sections $a_1$ et $a_s$ constituent les extrémités du mètre, tandis que pour tout $i \in \{2, \ldots, s-1\}$, la section $a_i$ est adjacente aux sections $a_{i-1}$ et $a_{i+1}$). La figure 2 représente un mètre pliant dans lequel toutes les sections ont la même taille. Le charpentier a également acheté un étui de longueur $B$ dans lequel il souhaite ranger son mètre plié. La question est : peut-on plier le mètre de façon à ce qu'il puisse être rangé dans cet étui ?

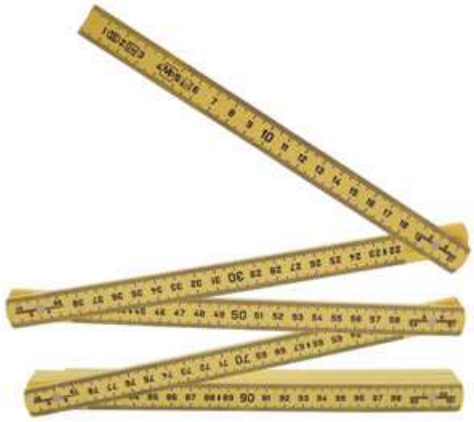


FIGURE 2 – Un mètre pliant ayant des sections de même longueurs.

**Q 13.1** Montrer que ce problème appartient à NP.

**Q 13.2** Montrer que ce problème est NP-complet. Vous pourrez supposer que le problème PARTITION est NP-complet.

*Rappel :* le problème PARTITION est le suivant :
**Entrée :** un ensemble de $n$ nombres entiers positifs $S = \{x_1, x_2, \ldots, x_n\}$.
**Question :** existe-t-il un sous-ensemble $S'$ de $S$ tel que $\sum_{x_i \in S'} x_i = \sum_{x_i \in S \setminus S'} x_i$ ?

## Exercice 14 Solitaire

**Q 14.1** On considère une version d'un jeu de solitaire qui se joue sur un damier de taille $m \times m$. Sur

chacune des $m^2$ cases se trouve soit une pierre bleue, soit une pierre rouge, soit rien du tout. On joue en retirant des pierres du damier jusqu'à ce que chaque colonne ne contienne que des pierres d'une seule couleur (ou pas de pierre du tout), et chaque ligne contienne au moins une pierre. On gagne si on atteint cet objectif. Selon la configuration de départ du damier, gagner peut être possible ou non. Le problème SOLITAIRE est le suivant : étant donnée une configuration de départ (un damier de taille $m \times m$, et la position et la couleur des pierres sur les cases), est-il possible de gagner ? Montrer que ce problème est NP-complet.
*Indication :* Réduire le problème 3-SAT à ce problème.

**Q 14.2** Aurait-on pu faire la même preuve en montrant que 2-SAT $\leq_{\mathrm{P}}$ SOLITAIRE ? Même question en montrant que SAT $\leq_{\mathrm{P}}$ SOLITAIRE ?

---

**Exercice 15 Tiré de l'examen réparti 1 en 2015-2016**

---

Soit $G = (V, A)$ un graphe **orienté**. On dit qu'un sous-ensemble de sommets $V'$ est :
- *stable* si aucun arc $(u, v) \in A$ n'a ses deux extrémités $u$ et $v$ dans $V'$ ;
- *dominant* si pour tout sommet $v \notin V'$, il existe $u \in V'$ tel que $(u, v) \in A$.

On dit qu'un sous-ensemble de sommets $V'$ est un *noyau* si $V'$ est à la fois stable et dominant. Par exemple, dans le graphe $G_1$ de la figure 3 l'ensemble $\{2, 5\}$ est un noyau.

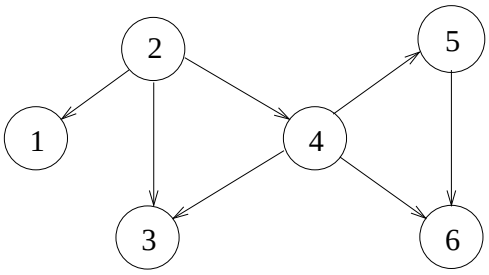



FIGURE 3 – Graphe $G_1$

**Q 15.1** Construire un graphe à 3 sommets qui n'admet pas de noyau.
Soit le problème NOYAU consistant à déterminer, étant donné un graphe orienté $G = (V, A)$, si $G$ admet un noyau. On veut montrer que ce problème est NP-complet par réduction du problème SAT.

**Q 15.2** Montrer que NOYAU $\in$ NP.

**Q 15.3** Soit $I$ une instance de SAT sur $n$ variables $\{x_1, x_2, \ldots, x_n\}$ et $m$ clauses $C_1, C_2, \ldots, C_m$. On considère le graphe $G(I)$ constitué de :
- pour chaque variable $x_i$, deux sommets $x_i$ et $\overline{x_i}$ avec les deux arcs $(x_i, \overline{x_i})$ et $(\overline{x_i}, x_i)$.
- Pour chaque clause $C_i$, trois sommets $a_i, b_i, c_i$ et trois arcs $(a_i, b_i)$, $(b_i, c_i)$ et $(c_i, a_i)$.
- Si $x_j$ apparaît dans $C_i$ alors l'arc $(x_j, a_i)$, si $\overline{x_j}$ apparaît dans $C_i$ alors l'arc $(\overline{x_j}, a_i)$.

On souhaite montrer que la transformation $t : I \to G(I)$ est une réduction polynomiale de SAT à NOYAU.

a) Combien $G(I)$ a-t-il de sommets ?
b) Soit $\sigma$ une valeur de vérité qui satisfait toutes les clauses de $I$ : montrer qu'alors $G(I)$ admet un noyau.

c) Réciproquement, soit $V'$ un noyau de $G(I)$. Montrer que pour tout $i \in \{1, \ldots, n\}$, $V'$ contient un et un seul sommet parmi $\{x_i, \overline{x_i}\}$. Montrer qu'il existe une valeur de vérité qui satisfait toutes les clauses de $I$.
d) Conclure.

**Q 15.4** Que pensez-vous du problème de l'existence de noyau dans un graphe non orienté (la définition du problème est celle donnée pour le cas orienté en changeant "arc" par "arête") ?

## Exercice 16 Tiré de l'examen réparti 1 (deuxième session) en 2015-2016

On considère dans cet exercice le problème qui consiste à savoir s'il existe entre deux sommets donnés d'un graphe un chemin élémentaire de longueur (exactement) $K$, où $K$ est une entrée du problème. Plus précisément, ce problème est le suivant :

*Entrée* : un graphe orienté valué $G = (S, A)$, deux sommets $u \in S$ et $v \in S$, et un entier $K$.
*Question* : existe-t-il un chemin élémentaire [1] de longueur égale à $K$ pour aller de $u$ à $v$ ?

Nous souhaitons montrer que ce problème, appelé CheminLongueurFixée est NP-complet.

**Q 16.1** Montrer que le problème CheminLongueurFixée appartient à $NP$.

**Q 16.2** On suppose que le problème suivant, appelé SubsetSum (pour "Somme de sous-ensembles"), est NP-complet.
*Entrée* : $n$ entiers positifs $S = \{v_1, \ldots, v_n\}$, et un entier positif $B$.
*Question* : existe-t-il un sous-ensemble $S'$ de $S$ tel que $\sum_{i \in S'} i = B$ ?

On souhaite réduire ce problème au problème CheminLongueurFixée. La réduction est la suivante :

Soit $\{v_1, \ldots, v_n, B\}$ une instance de SubsetSum. On construit l'instance de CheminLongueurFixée de la manière suivante (voir figure ci-dessous) :
— $S = \bigcup_{1 \leq i \leq n} \{x_i, z_i\} \cup \{x_0\}$
— Pour tout $i \in \{1, \ldots, n\}$, il existe des arcs $(x_{i-1}, z_i)$ et $(z_i, x_i)$ pondérés par 0 et un arc $(x_{i-1}, x_i)$ pondéré par $v_i$.
On cherche à décider si, dans ce graphe, il existe un chemin de $x_0$ à $x_n$ de longueur $B$.

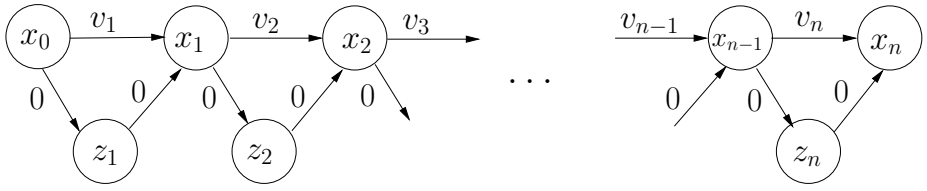



Figure 4 – Instance de CheminLongueurFixée correspondant à une instance $\{v_1, \ldots, v_n, B\}$ de SubsetSum.

Montrer que la réponse à une instance de SubsetSum est "oui" si et seulement si la réponse à l'instance correspondante du problème CheminLongueurFixée est "oui".

**Q 16.3** Montrer que CheminLongueurFixée est NP-complet.

1. Un *chemin élémentaire* est un chemin ne passant pas deux fois par un même sommet, c'est-à-dire dont tous les sommets sont distincts. La *longueur* d'un chemin $c$ est la somme des valeurs des arêtes de $c$.

## Exercice 17 Clique

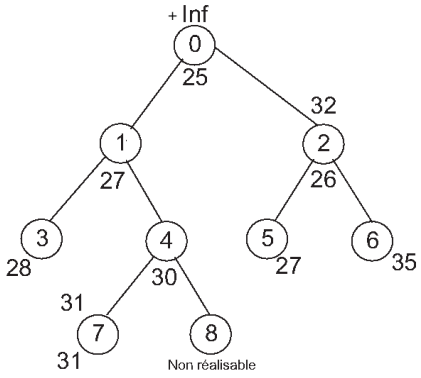
**Q 17.1** Montrer que si P=NP, alors il existe un algorithme polynomial qui prend en entrée un graphe non orienté $G = (S, A)$ et qui retourne une plus grande clique de $G$.

Remarque : on demande ici un algorithme qui résoud un problème d'optimisation (retourner une clique la plus grande possible de $G$). L'hypothèse P=NP implique que le problème Clique appartient à P. On souhaite donc montrer que s'il est possible de tester en polynomial si $G$ contient une clique de taille $k$, alors il est également possible de retourner en temps polynomial une clique de taille maximale.

# 2 Méthodes arborescentes

## Exercice 18 Un premier exemple

On considère l'arbre d'énumération partiel suivant pour un problème de minimisation :




La valeur inscrite à l'intérieur de chaque cercle est le numéro du sommet. La valeur en-dessous (respectivement au-dessus) de chaque cercle correspond à une borne inférieure (respectivement supérieure) de la valeur optimale dans la branche correspondante.

**Q 18.1** Déterminer la meilleure borne supérieure de la valeur optimale du problème.

**Q 18.2** Quelles sont les feuilles de l'arbre qui peuvent être élaguées ?

**Q 18.3** Quelles sont les feuilles de l'arbre qui doivent être séparées ?

## Exercice 19 Sac-à-dos

On considère le problème du Sac à dos dont l'énoncé peut-être décrit par : "Étant donné un ensemble de $n$ objets possédant chacun un poids $p_i$ et une valeur (utilité) $u_i$ et étant donné un poids maximum $b$ pour le sac, quels objets faut-il mettre dans le sac de manière à maximiser la valeur totale sans dépasser le poids maximal autorisé pour le sac ?"

**Q 19.1** Proposer une formulation linéaire en nombres entiers du problème du Sac à dos (les variables de décision seront notées $x_j$ pour $j = 1, \ldots, n$).

**Q 19.2** Proposer un algorithme glouton pour ce problème. Appliquer cet algorithme sur l'instance suivante, avec $b = 20$ :

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $p_i$ | 14 | 10 | 8 | 6 | 5 | 2 |
| $u_i$ | 24 | 19 | 16 | 13 | 5 | 3 |

Table 1 – Instance du problème de sac-à-dos

**Q 19.3** Adapter l'algorithme glouton au cas de la relaxation continue, et montrer que la solution (continue) obtenue est alors optimale. On pourra supposer pour simplifier que tous les ratios $u_i/p_i$ sont différents.

Trouver alors l'optimum continu de l'instance de la question précédente.

**Q 19.4** A partir de la question précédente, proposer des bornes supérieure et inférieure de la valeur d'une solution optimale du problème du Sac à dos.

**Q 19.5** Appliquer alors un algorithme de séparation et évaluation sur l'exemple de la question 2. On commencera par brancher sur les objets de plus grands poids, en considérant d'abord qu'on les prend, puis qu'on ne les prend pas.

**Q 19.6** Comparer le nombre de nœuds explorés et le nombre de nœuds de l'arbre complet.

# 3 Algorithmes d'approximation

## Exercice 20 Sac à dos

On considère une version simplifiée du problème de sac à dos. Le problème est le suivant : étant donnés $n$ entiers positifs $A = \{a_1, \ldots, a_n\}$ et un entier positif $B$, le but est de trouver un sous-ensemble $S$ de $A$ tel que la somme des ses éléments, $\sum_{a_i \in S} a_i$, soit aussi grande que possible tout en ne dépassant pas $B$.

**Q 20.1** On considère l'algorithme suivant :
$S = \emptyset$
$Poids = 0$
**Pour** $i$ allant de 1 à $n$ **faire**
  Si $Poids + a_i \leq B$ alors
    $S := S \cup \{a_i\}$
    $Poids := Poids + a_i$
**Fin pour**

Donner une instance telle que le rapport entre le poids de l'ensemble retourné par l'algorithme et le poids de l'ensemble optimal est aussi petit que possible.

**Q 20.2** Que peut-on dire du rapport d'approximation de cet algorithme si l'on classe auparavant les entiers de $A$ par ordre décroissant ($a_1 \geq \cdots \geq a_n$) ? Quelle est dans ce cas la complexité de l'algorithme ?

**Q 20.3** Modifier légèrement l'algorithme donné dans la première question afin de le rendre $\frac{1}{2}$-approché. Votre algorithme doit s'éxécuter en $O(n)$.

## Exercice 21 Tiré de l'examen réparti 1 2014-2015

Etant donné un graphe non orienté $G = (V, E)$, une *couverture* de ce graphe est un sous-ensemble de sommets $V'$ tel que toute arête a au moins une extrémité dans $V'$ :

$$\forall (i, j) \in E, i \in V' \text{ ou } j \in V'$$

Par exemple, sur le graphe de la figure 5, $V' = \{1, 3, 4, 6\}$ est une couverture, mais $V'' = \{3, 6, 7\}$ n'en est pas une car l'arête $(4, 5)$ n'est pas *couverte*.
On considère alors le problème suivant : étant donné un graphe $G = (V, E)$, trouver une couverture de $G$ de taille minimale.

**Q 21.1** Donner une solution optimale sur le graphe de la figure 5 (on ne demande pas de justification).

**Q 21.2** De manière générale, dans un graphe à $n$ sommets, donner un majorant du nombre de solutions réalisables.

**Q 21.3** Montrer que le problème de décision associé (étant donnés un graphe $G$ et un entier $k$, déterminer s'il existe une couverture de taille au plus $k$) est NP-complet. On suppose que l'on sait que le problème STABLE est NP-complet.

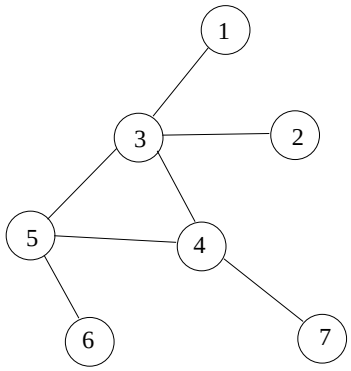



FIGURE 5 – Exemple

**Q 21.4** On considère l'algorithme $APPROX_{VC}$ suivant :

———— $APPROX_{VC}$ ————————————————————

$C \leftarrow \emptyset$
Tant qu'il existe dans $G$ une arête $e = (i, j)$ non couverte par $C$, faire :
$C \leftarrow C \cup \{i, j\}$
Fin Tant Que
Renvoyer $C$

———————————————————————————————

Appliquer l'algorithme sur le graphe de la figure 5. On donnera simplement à chaque étape l'arête dont les deux extrémités sont ajoutées, ainsi que la solution renvoyée.

**Q 21.5** Montrer que $APPROX_{VC}$ est 2-approché.

**Q 21.6** Trouver un graphe $G$ et une exécution de $APPROX_{VC}$ où la solution renvoyée $C$ est telle que $|C| = 2OPT(G)$.

**Q 21.7** On considère maintenant le problème de la couverture pondérée où chaque sommet $v$ a un poids $w(v)$, et l'on cherche une couverture $V'$ de poids total $\sum_{v \in V'} w(v)$ minimal. $APPROX_{VC}$ est-il toujours 2-approché pour le problème de la couverture pondérée ?

**Q 21.8** On considère le problème du stable maximum et l'on propose l'algorithme $APPROX_{Stable}$ consistant à renvoyer $V \setminus APPROX_{VC}(G)$.
Cet algorithme renvoie-t-il toujours une solution réalisable ? Est-il $\frac{1}{2}$-approché ?

## Exercice 22 Ordonnancement de tâches

On se donne un ensemble $S$ de $n$ tâches $\{1, \ldots, n\}$, chaque tâche $i \in S$ ayant une durée (temps d'exécution) $l_i$, à ordonnancer sur $m$ machines. On considère l'algorithme suivant : les tâches sont triées par durées croissantes, et dès qu'une machine est disponible alors la plus petite tâche non encore ordonnancée est exécutée sur cette machine. Cet algorithme s'appelle SPT (pour "shortest processing time first"). Nous avons vu dans l'exercice 7 que cet algorithme minimise la date de fin moyenne des tâches. Nous considérons maintenant une autre fonction objectif, qui consiste à minimiser la date à laquelle toutes les tâches ont été exécutées. Ainsi, si on note $C_i$ la date de fin d'exécution de la tâche $i$, alors on cherche à minimiser $C_{max} = \max_{i \in \{1,\ldots,n\}} C_i$ ($C_{max}$ est appelée la date de fin de l'ordonnancement).

**Q 22.1** Exécuter cet algorithme sur l'instance suivante : 2 machines et 3 tâches de durées 1,1 et 2.

Quelle est la date de fin de l'ordonnancement obtenu ? Quelle est la date de fin de l'ordonnancement optimal ?

**Q 22.2** On considère maintenant une instance quelconque $I$. Soit $OPT$ la durée d'un ordonnancement optimal de $I$. En remarquant que $\frac{1}{m}\sum_{i=1}^{n} l_i \leq OPT$ et $\max_{i\in\{1,\ldots,n\}} l_i \leq OPT$ (expliquer pourquoi), montrer que cet algorithme est $(2-\frac{1}{m})$-approché.

**Q 22.3** Déterminer une instance critique pour cet algorithme.

---

**Exercice 23 Tiré de l'examen réparti 1 en 2015-2016**

---

On considère le problème consistant à minimiser le nombre de boîtes de taille 1 utilisées pour placer $n$ objets de taille $\{a_1,\ldots,a_n\}$, avec, pour tout $i \in \{1,\ldots,n\}$, $0 < a_i \leq 1$. La taille cumulée de l'ensemble des objets placés dans une même boîte ne doit pas dépasser 1. Ce problème d'optimisation est connu sous le nom de BinPacking.

L'algorithme ProchaineBoîte, décrit ci-dessous, retourne une solution approchée avec garantie de performance pour ce problème.

**Entrées** : $n$ nombres $\{a_1,\ldots,a_n\}$ tels que, pour tout $i \in \{1,\ldots,n\}$, $0 < a_i \leq 1$
**Sorties** : Un nombre de boîtes de taille 1 permettant de placer les nombres $a_i$.
boîte_courante $\leftarrow 1$
taille_cumulée_boîte_courante $= 0$
**pour** *i allant de 1 à n* **faire**
  **si** *taille_cumulée_boîte_courante* $+ a_i \leq 1$ **alors**
    //$a_i$ rentre dans la boîte courante : on l'y place
    taille_cumulée_boîte_courante $\leftarrow$ taille_cumulée_boîte_courante $+ a_i$
  **sinon**
    //$a_i$ ne rentre pas dans la boîte courante : on le met dans une nouvelle boîte
    boîte_courante $\leftarrow$ boîte_courante $+ 1$
    taille_cumulée_boîte_courante $\leftarrow a_i$
**retourner** boîte_courante

**Algorithme 1 :** Algorithme ProchaineBoîte

**Q 23.1** Exécuter l'algorithme ProchaineBoîte sur l'instance $\{0.9, 0.3, 0.1, 0.8, 0.2\}$.
*On donnera simplement la solution obtenue (nombre de boîtes et liste des objets dans chaque boîte).*

**Q 23.2** Soit $I$ une instance de notre problème, soit $OPT$ le nombre de boîtes retournées dans une solution optimale pour $I$, et soit $x$ le nombre de boîtes retournées par l'algorithme ProchaineBoîte sur l'instance $I$. Nous souhaitons déterminer le rapport d'approximation de l'algorithme ProchaineBoîte. Notons $q_i$ la taille cumulée des objets placés dans la boîte $i$ par ProchaineBoîte.

a) On suppose que $x$ est pair. Expliquer pourquoi $q_1 + q_2 > 1$, puis montrer que $\sum_{i=1}^{x} q_i > \frac{x}{2}$. Montrer alors que $OPT > \frac{x}{2}$.
b) On suppose que $x$ est impair : $x = 2y + 1$. Montrer que $OPT \geq y + 1$.
c) Déduire des questions précédentes un rapport d'approximation de l'algorithme ProchaineBoîte.

**Q 23.3** Appliquer l'algorithme ProchaineBoîte sur l'instance $I_p = \{1, \frac{1}{p}, 1, \frac{1}{p}, 1, \frac{1}{p}, \ldots, 1, \frac{1}{p}\}$ où l'on répète $p$ fois le motif $\{1, \frac{1}{p}\}$. Donner la valeur optimale sur $I_p$, et conclure quant au rapport d'approximation de l'algorithme ProchaineBoîte.

**Q 23.4** Soit $\varepsilon > 0$. Montrer qu'il n'existe pas d'algorithme polynomial $(\frac{3}{2} - \varepsilon)$-approché pour notre problème d'optimisation, sauf si $P = NP$.
*Indication :* on pourra montrer que si un tel algorithme existait il permettrait de résoudre le problème Partition.

# 4 TME des semaines 3 à 5

### Cadre applicatif

De plus en plus de produits sont équipés d'une ou de plusieurs formes de technologie d'identification automatique telle que l'identification par radio-fréquence (RFID). Cette technologie permet notamment la collecte de données, le stockage et la transmission de l'information produite tout au long du cycle de vie du produit. Pour un produit ne possédant qu'une étiquette RFID, toutes les informations pertinentes disponibles devraient idéalement être stockées sur l'étiquette RFID du produit. Toutefois, en raison de la capacité mémoire limitée des étiquettes RFID, les utilisateurs doivent être sélectifs dans l'allocation des données aux étiquettes. Dans ce projet, on s'intéresse au problème de collecte de données liées à un produit équipée d'une étiquette RFID en supposant qu'une utilité est associée à chaque donnée et que la capacité de l'étiquette RFID est limitée. L'objectif est de maximiser l'utilité globale de l'ensemble des données disponibles sur l'étiquette RFID du produit.

### Modèle

Ce problème se modélise comme un problème de Sac-à-dos dont l'énoncé peut-être décrit par : "Étant donné un ensemble de $n$ objets possédant chacun un poids $p_i \in \mathbb{N}$ et une valeur (utilité) $u_i \in \mathbb{N}$, et étant donné un poids maximum $b \in \mathbb{N}$ pour le sac, quels objets faut-il mettre dans le sac de manière à maximiser la valeur totale des objets sélectionnés, et ce sans dépasser le poids maximal autorisé pour le sac ?"

Le problème peut donc s'écrire de la façon suivante :

Maximiser $\sum_{i=1}^{n} x_i u_i$

sous les contraintes :
$\sum_{i=1}^{n} x_i p_i \leq b$
$x_i \in \{0, 1\}$ pour tout $i \in \{1, \ldots, n\}$

Le but de ce TP est d'implémenter différents algorithmes, exacts et approchés, pour résoudre le problème du sac-à-dos, et de les tester expérimentalement sur différentes classes d'instances. Le choix du langage de programmation est libre.

### Partie 1

1. Coder l'algorithme glouton vu en TD qui sélectionne les objets dans l'ordre décroissant de leur bénéfice (rapport $\frac{u_i}{p_i}$) : quand un objet rentre dans le sac-à-dos, on le sélectionne et on considère l'objet suivant, sinon on ne le sélectionne pas et on considère l'objet suivant.
2. Pour produire des courbes qui synthétisent les tests que vous aurez effectués dans cette question et dans les questions suivantes, vous utiliserez un outil de tracé de courbes (pour information, une annexe à ce TP indique quelques commandes de base pour l'utilisation de l'outil de tracé de courbes Gnuplot). Les instances étant générées de façon aléatoire, il est important que pour chaque jeu de paramètres caractérisant une instance, plusieurs instances soient générées.

   Pour des instances non corrélées (voir section "Instances"), tracez une courbe indiquant le temps de calcul de votre algorithme en fonction du nombre d'objets de l'instance.

**Partie 2**

1. Mettre en œuvre une méthode arborescente exacte pour résoudre le problème du sac-à-dos. Vous n'utiliserez pour l'instant pas de bornes pour élaguer votre arbre (mais vous pouvez d'ores et déjà penser qu'il y en aura !). Chaque feuille de l'arbre correspondra à un sous-ensemble d'objets (une solution réalisable ou non du problème du SAC-À-DOS), et tout l'arbre sera donc parcouru. Vous mémoriserez la meilleure solution réalisable rencontrée.

   *Indication :* Vous pourrez utiliser la méthode de branchement suivante : le noeud racine (niveau 0) ne contient aucun objet ; il est père de deux noeuds de niveau 1 : celui de gauche contient l'objet 1, et celui de droite ne le contient pas. Pour chaque noeud de niveau $i$, vous créerez deux noeuds : celui de gauche contiendra, en plus des objets courants l'objet $i$, et celui de droite ne le contiendra pas.

   Vous pourrez implémenter cet algorithme en utilisant une pile (qui au départ ne contient que le noeud racine) : tant que la pile n'est pas vide, vous considérez le noeud courant de la pile, $n_{courant}$. Vous retirez ce noeud de la pile, après éventuellement avoir mis à jour la meilleure solution rencontrée. Vous créerez aussi, si le noeud n'est pas une feuille (i.e. s'il n'est pas de niveau $n-1$), ses deux fils que vous ajouterez à la pile.

   **Testez votre méthode** sur des petites instances de 3 ou 4 objets pour voir si cela fonctionne (parcours complet de l'arbre, meilleure solution calculée).

2. S'il y a $n$ objets dans le sac à dos, combien l'arbre parcouru contiendra-t-il de feuilles, de noeuds ? Testez votre méthode sur de petites instances et vérifiez que ces valeurs sont correctes pour les instances testées.

   Jusqu'à quelle valeur $N_{max}$ de $n$ votre méthode fonctionne-t-elle rapidement (de l'ordre de la seconde) ?

3. À partir de l'exercice 19 vu en TD, proposez des bornes supérieure et inférieure de la valeur d'une solution optimale du problème du SAC-À-DOS. Expliquez comment utiliser ces bornes pour pouvoir élaguer certaines branches de l'arbre de recherche. Ajoutez cette possibilité d'élagage au parcours de l'arbre de recherche effectué dans les questions précédentes. Vous obtiendrez ainsi un algorithme de type "branch and bound".

   **Testez** votre algorithme (*branch and bound*) sur l'instance de l'exercice 19. Vous vérifierez la concordance des noeuds visités et de la solution trouvée.

   Testez votre algorithme sur une instance de taille $N_{max}$ et comparez avec la question précédente.

4. Il s'agit dans cette question d'évaluer la performance de votre algorithme, en terme de temps de calcul ou de nombre de noeuds parcourus.

a) Pour différentes classes d'instances (voir section "Instances"), tracez une courbe indiquant le temps de calcul de votre algorithme de branch and bound en fonction du nombre d'objets de l'instance. Vous tracerez une courbe par classe d'instance.

b) Pour différentes classes d'instances, tracez une courbe indiquant le nombre de noeuds visités (et/ou la proportion de noeuds visités) par votre algorithme de branch and bound en fonction du nombre d'objets de l'instance.

c) (question facultative) On voudrait mesurer l'influence de l'ordre dans lequel l'algorithme considère les objets (le coût d'un prétraitement est très faible pour un gain potentiellement intéressant). Comparer le temps d'exécution de votre algorithme :
   — si les objets sont considérés dans un ordre quelconque
   — si les objets sont considérés dans l'ordre croissant de leur poids
   — si les objets sont considérés dans l'ordre croissant de leur bénéfice (rapport $\frac{u_i}{p_i}$).

### Partie 3

1. Codez l'algorithme pseudo-polynomial vu en cours qui résout de manière exacte le problème du Sac-à-dos.
   Remarque : Vous pourrez, si vous le souhaitez, améliorer la borne supérieure (fixée à $nV_{max}$) vue en cours.
2. Testez l'algorithme implémenté dans la question précédente avec $n = 100$ objets, avec des objets ayant des valeurs comprises entre 0 et $M$. Indiquez jusqu'à quelle valeur de $M$ votre programme peut être utilisé. Vous pourrez tracer une courbe indiquant la variation du temps de calcul en fonction de $M$.
3. Codez le schéma d'approximation fortement polynomial vu en cours.
4. (question facultative) Testez cet algorithme pour différentes valeurs de $\varepsilon$ sur des instances de $n = 100$ objets par exemple. Pour chaque valeur de $\varepsilon$ testée, indiquer le temps de calcul de l'algorithme : vous pourrez à cet effet tracer une courbe indiquant la variation du temps de calcul en fonction de $\varepsilon$, pour $M$ fixé. Vous pourrez faire cette étude du compromis entre $\varepsilon$ et le temps de calcul pour différentes valeurs de $M$.
5. (question facultative) Pour des instances pour lesquelles l'utilisation d'un algorithme exact est possible (branch and bound ou algorithme pseudo-polynomial), comparer la qualité des solutions retournées par le schéma d'approximation à celles retournées par l'algorithme exact. Il sera à cet effet utile de tracer une courbe indiquant l'erreur commise en fonction de la valeur de $\varepsilon$.

### Instances

Afin de tester vos algorithmes, différentes classes d'instances seront considérées.
Chaque classe sera testée pour différentes valeurs d'un paramètre $R = 100, 1000, 10000$, et pour plusieurs tailles d'instances. Si votre méthode fonctionne en quelques secondes jusqu'à une taille approximative $T_{max}$ (par exemple $T_{max} = 1000$), vous regarderez le temps de calcul pour des tailles d'instances $T_{max}/10, 2T_{max}/10, 3T_{max}/10, \ldots, T_{max}$ (100, 200, 300,...,1000 sur l'exemple). La capacité du sac-à-dos est fixée à $b = \frac{1}{2}\sum_{i=1}^{n} p_i$.
Les profits sont définis en fonction des poids. Différentes classes d'instances sont ainsi générées. Elles sont définies de la manière suivante :

— Données non-corrélées : $p_j$ et $u_j$ sont choisis aléatoirement selon une loi uniforme $U[1, R]$. Cette classe d'instances illustre le cas où le profit ne dépend pas poids.
— Données faiblement corrélées : $p_j$ est choisi aléatoirement selon une loi uniforme $U[1, R]$ et les utilités sont choisis aléatoirement selon une loi uniforme $U[p_j - \frac{R}{10}, p_j + \frac{R}{10}]$ de façon à ce que $u_j \geq 1$.
— Données "subset sum" : les poids $p_j$ suivent une loi uniforme $U[1, R]$ et $u_j = p_j$.

Si vous le souhaitez, vous pouvez également utiliser des instances réputées difficiles, accessibles à l'URL : `http://www.diku.dk/hjemmesider/ansatte/pisinger/codes.html`

**Annexe** : Créer des courbes avec Gnuplot

Gnuplot peut être installé sous Linux ou Windows. Deux modes d'utilisation sont possibles. Un *mode interactif* dans lequel on est face au terminal Gnuplot et il faut alors lancer les commandes les unes à la suite des autres pour obtenir le graphique souhaité, et un *mode script* dans lequel on écrit un script Gnuplot qui contient toutes les commandes qu'il faut exécuter, ce script sera alors lancé.

Vous trouverez ci-dessous un bref descriptif illustré par un exemple des commandes de base pour produire une courbe en mode interactif.
Une documentation complète est accessible à `http://www.gnuplot.info/docs_4.6/gnuplot.pdf` et des démos à `http://gnuplot.sourceforge.net/demo/`.

L'utilisation de Gnuplot en mode interactif se fait en ligne de commande dont le prompt est : `gnuplot>`.
La commande PLOT permet de tracer des courbes en 2D. Sa syntaxe exacte est :

```
    PLOT {ranges} { | {"" {using...}}}
                    {title} {style} {,  {title} {style}...}
{} indique un argument optionnel
 |  indique un choix
```

L'option `using` permet notamment de faire un choix sur les colonnes à utiliser pour tracer une courbe.
L'option `range` spécifie le domaine qui va être affiché sur la courbe.
L'option `style` permet d'établir un choix sur le style utilisé parmi `boxes`, `dots`, `errorbars`, `impulses`, `lines`, `linespoints`, `points` et `steps`.
Le titre de chaque courbe apparaît sur le graphique. Par défaut ce titre est le nom de la fonction ou le nom du fichier de données. Ce nom peut être changé en utilisant l'option `title`.

**Exemple** : Soit le fichier 'test.dat' dont le contenu est :

```
#Colonne 1: Taille des instances
#Colonne 2: Valeur du Gap
100            0
150            1
200            2
250            2.3
300            5
350            8
400            8.4
450            17
```

Les lignes de commande ci-dessous :

```
gnuplot> set xlabel 'Taille'
gnuplot> set ylabel 'Valeur'
gnuplot> plot "test.dat" using 1:2 with lines title 'Gaps obtenus par A'
```

permettent d'obtenir le graphique suivant :

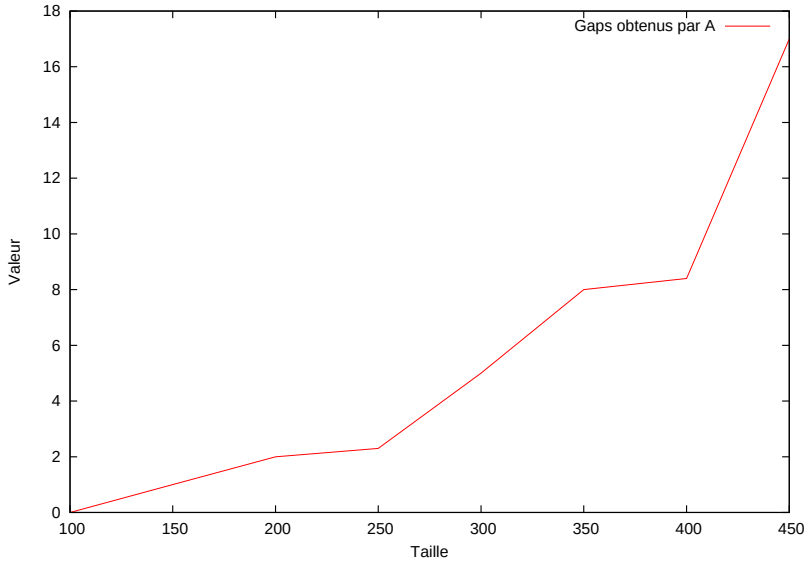



FIG. 6: Exemple Gnuplot

L'un des principaux intérêt de Gnuplot est qu'il peut utiliser des fichiers de données au format texte pour produire une courbe. Pour obtenir un fichier de sortie (courbe) 'test.ps' au format postscript, il suffit de lancer les lignes de commande suivantes :

```
gnuplot> set term postscript eps color
gnuplot> set output "test.ps"
gnuplot> set xlabel 'Taille'
gnuplot> set ylabel 'Valeur'
gnuplot> plot "test.dat" using 1:2 with lines title 'Gaps obtenus par A'
```

Ce fichier vous permettra notamment d'insérer la figure au format postscript dans un rapport.

# 5 Annales

**UE COMPLEX.**
**M1 Informatique.**

**Examen du 3 novembre 2016. Durée : 2 heures**

*Une feuille recto-verso est autorisée, tout autre document est interdit.*
*Téléphones portables éteints et rangés dans vos sacs.*

*Le barème est indicatif et est susceptible d'être modifié.*

## **Exercice** 1 (5 points)

Dans chacun des cas suivants dites si les affirmations sont vraies ou fausses. Aucune justification n'est demandée.

*1 point par bonne reponse, -0.5 par mauvaise réponse (0 si pas de réponse).*

1. Soit $A$ et $B$ deux problèmes de $NP$. Si $A$ se réduit polynomialement à $B$ et $B$ à $A$, alors $A$ et $B$ sont tous les deux $NP$-complets.
2. Soit $A$ un problème $NP$-complet, et $B$ un problème de $NP$ qui n'est pas $NP$-complet. Alors $B$ se réduit polynomialement à $A$.
3. Pour le problème du sac-à-dos, une méthode de *branch and bound* permet de n'explorer qu'un nombre de noeuds de l'arbre de recherche polynomial en fonction de la taille de l'instance.
4. Pour un problème de minimisation, un algorithme (polynomial) 2-approché renvoie sur toute instance $I$ une solution de valeur égale à exactement deux fois la valeur optimale de $I$.
5. Si un langage est décidable par une machine de Turing non déterministe, alors il est décidable par une machine de Turing déterministe.

## **Exercice** 2 (5 points)

Soit $X = \{x_1, x_2, \ldots, x_n\}$ un ensemble de variables binaires, et $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ un ensemble de clauses sur ces variables.

**Question** 1 (0.5/5) — Supposons que les clauses ne contiennent que des littéraux négatifs (que des négations $\overline{x_i}$), par exemple $C_1 = (\overline{x_1} \vee \overline{x_3} \vee \overline{x_5})$,... . Comment peut-on satisfaire toutes les clauses ?

**Question** 2 (4.5/5) — On s'interroge alors maintenant sur la possibilité de satisfaire toutes les clauses en mettant (au moins) $k$ variables à vrai. Nous définissons ainsi le problème SatNegatif, dans lequel une instance est la donnée de $n$ variables, de $m$ clauses ne contenant que des littéraux négatifs, et d'un entier $k$. La question est de savoir s'il existe une valeur de vérité ayant au moins $k$ variables vraies et satisfaisant toutes les clauses.

1. Ce problème est-il dans NP ?
2. On rappelle la définition du problème Stable : Dans un graphe (non orienté) $G = (V, E)$, un stable est un ensemble de sommets deux à deux non adjacents. Etant donné un graphe $G$ et un entier $p$, la question est de savoir s'il existe un stable de taille au moins $p$ ou pas.

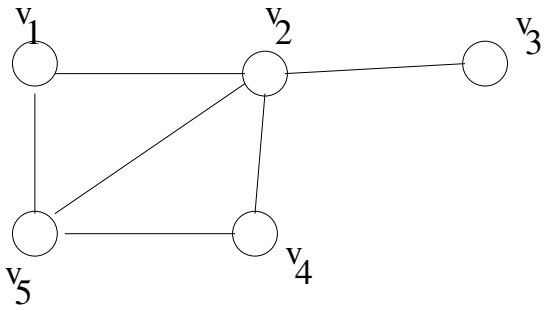
   On suppose connu le fait que ce problème est NP-complet.

   Soit un graphe $G = (V, E)$ et un entier $p$, $V = \{v_1, v_2, \ldots, v_n\}$ étant l'ensemble des $n$ sommets du graphe et $E = \{e_1, e_2, \ldots, e_m\}$ l'ensemble des $m$ arêtes du graphe.

   On considère l'instance $f(G, k)$ de SatNégatif suivante :

— Pour chaque sommet $v_i$ il y a une variable $x_i$ ;
— Pour chaque arête $e_j$ reliant les sommets $v_i$ et $v_\ell$ il y a une clause $C_j = (\overline{x_i} \vee \overline{x_\ell})$
— $k = p$.

(a) Soit le graphe $H$ de la figure suivante, et $k = 3$. Construire l'instance $f(H, k)$. Quelles sont les réponses à $(H, k)$ et à $f(H, k)$ ?




(b) De manière générale, montrer que la réponse à une instance $(G, k)$ de Stable est oui si et seulement si la réponse à l'instance $f(G, k)$ de SatNégatif est oui.

(c) Conclure sur la complexité du problème SatNegatif.

**Exercice** 3 (6 points)

On considère le problème Max Sat où, étant donné $n$ variables binaires $x_1, x_2, \ldots, x_n$ et $m$ clauses $C_1, \ldots, C_m$, le but est de trouver une valeur de vérité satisfaisant un nombre maximum de clauses.

**Question** 1 (1.5/6) — On considère l'algorithme `A` consistant à prendre la meilleure des deux solutions $\sigma_1$ et $\sigma_2$, $\sigma_1$ consistant à mettre toutes les variables à vrai et $\sigma_2$ consistant à mettre toutes les variables à faux. Quel est le rapport d'approximation de cet algorithme ?

**Question** 2 (2.5/6) —

On considère dans cette question l'instance suivante : $X = \{x_1, x_2, x_3, x_4\}$, $C_1 = (x_1 \vee \overline{x_3} \vee x_4)$, $C_2 = (\overline{x_1} \vee \overline{x_2})$, $C_3 = (x_2 \vee x_4)$, $C_4 = (\overline{x_1} \vee x_3 \vee \overline{x_4})$.

On cherche à construire un algorithme de *branch and bound*. L'arbre consiste à fixer $x_1$ à vrai ou à faux, puis $x_2$ à vrai ou à faux,...

1. Quelle est le nombre de noeuds d'un arbre complet (sans élagage) ?
2. Appliquer l'algorithme `A`, en donnant les clauses satisfaites (et leur nombre) par $\sigma_1$ et $\sigma_2$.
3. Considérons le noeud où l'on a mis $x_1$ et $x_2$ à vrai. En remplaçant $x_1$ et $x_2$ par leurs valeurs (vrai) dans les clauses, qu'obtient-on ? Est-il utile de poursuivre l'exploration à partir de ce noeud ?

**Question** 3 (2/6) — Formaliser l'approche précédente sous la forme d'un algorithme de *branch and bound*. On spécifiera le calcul des bornes inférieure et supérieure, ainsi que la (les) condition(s) d'élagage.

**Exercice** 4 (4 points)

Soit $G = (V, E)$ un graphe (non orienté). Un ensemble d'arêtes $E' \subset E$ est dit dominant si toute arête $e \notin E'$ a au moins une extrémité commune avec (au moins) une arête de $E$. Le problème AreteDomination consiste à déterminer un ensemble d'arêtes dominant de taille minimum.

On considère l'algorithme glouton suivant : on initialise $E' \leftarrow \emptyset$, et tant qu'il existe une arête $e \notin E'$ qui n'a aucune extrémité commune avec les arêtes de $E'$, on ajoute $e$ à $E'$.

**Question** 1 (0.5/4) — Montrer que l'algorithme renvoie toujours un ensemble d'arêtes dominant.

**Question** 2 (2.5/4) — Montrer que l'algorithme est 2-approché.

**Question** 3 (1/4) — Trouver une instance où l'algorithme renvoie une solution de taille exactement 2 fois la taille d'une solution optimale.

**Devinette** pour ceux qui s'ennuieraient (question non notée) : Comment faire 24 en utilisant une et une seule fois chacun des nombres 1, 3, 4 et 6 - en utilisant les opérations usuelles addition, soustraction, multiplication, division ? (par exemple 4× 6=24 mais on n'a pas utilisé 1 et 3, 3×6+4+1 est OK mais égale 23...)

**UE COMPLEX.**
**M1 Informatique.**

**Examen du 8 juin 2017. Durée : 2 heures**

*Documents non autorisés. Seule une feuille A4 portant sur les cours et les TD est autorisée.*
*Téléphones portables éteints et rangés dans vos sacs.*

*Le barème est indicatif et est susceptible d'être modifié.*
*Toutes les réponses doivent être justifiées.*

## **Exercice** 1 (10 points)

**Le problème des transferts**

Vous partez avec un certain nombre d'ami(e)s à Barcelone pour le week-end. A la fin du week-end, vous faîtes les comptes afin d'équilibrer les dépenses. Certains ont beaucoup dépensé et doivent donc recevoir de l'argent de ceux qui ont moins dépensé. On cherche maintenant à faire des transferts, de manière à épurer les comptes : un transfert est une somme d'argent qu'une personne donne à une autre.

Il faut bien sûr qu'à la fin tout le monde reçoive/verse ce qu'il faut. Afin que la procédure soit la plus simple possible, on cherche à minimiser le nombre de transferts.

> **Exemple explicatif :** supposons qu'il y ait 5 personnes $A, B, C, D, E$. La personne $A$ doit recevoir 2 euros, $B$ doit recevoir 3 euros, $C$ doit rembourser 1 euro, $D$ et $E$ doivent rembourser 2 euros.
> Une possibilité consiste à faire 4 transferts : $C \to A : 1$ (1 euro de $C$ vers $A$), $D \to A : 1$, $D \to B : 1$, et $E \to B : 2$.
> Une meilleure solution consiste à ne faire que 3 transferts : $D \to A : 2$, $C \to B : 1$, et $E \to B : 2$. Il s'agit ici de la solution optimale, aucune solution ne permet de faire moins de transferts.

Note : pour simplifier, nous supposerons dans tout l'exercice (1) qu'un transfert est forcément une somme d'une personne déficitaire vers une personne bénéficiaire et (2) que toute personne doit recevoir ou donner quelque chose, il n'y pas de personne à l'équilibre initialement.

**Question** 1 (3.5/10) —

On considère l'algorithme suivant :

> Tant que les comptes ne sont pas à l'équilibre :
> — parmi les personnes qui doivent (encore) verser de l'argent, considérer la personne $i$ qui doit le moins d'argent, soit $d_i$ la somme due.
> — parmi les personnes qui doivent (encore) recevoir de l'argent, considérer la personne $j$ qui doit recevoir le moins d'argent, soit $b_j$ la somme à recevoir.
> — Transférer $\min\{d_i, b_j\}$ euros de $i$ à $j$.
> — Mettre à jour les dettes/crédits de $i$ et $j$.

Appliquer l'algorithme sur l'exemple suivant :

| Bénéficiaires | Elena | Thibaut | Mathieu |
|---|---|---|---|
| montant à recevoir | 70 | 45 | 35 |

| Déficitaires | Stéphanie | Xavier | Bertrand | Gertrude |
|---|---|---|---|---|
| montant à donner | 60 | 40 | 30 | 20 |

On donnera simplement la liste des transferts sous la forme $i \to j : p$ (transfert de $p$ euros de $i$ à $j$.

Combien y a-t-il de transferts ? Est-ce la solution optimale ?

**Question** 2 (1.5/10) — De manière générale, s'il y a $n$ personnes, majorer le nombre de transferts de l'algorithme en fonction de $n$.

**Question** 3 (1.5/10) — Minorer, en fonction de $n$, le nombre de transferts de toute solution réalisable. En déduire que l'algorithme est 2-approché.

**Question** 4 (3.5/10) — On rappelle la définition du problème Partition, que l'on sait être NP-complet :

> Entrée : $n$ nombres entiers positifs $a_1, \ldots, a_n$, dont la somme est égale à $B$.
> Question : Existe-t-il $S \subseteq \{1, 2, \ldots, n\}$ tel que $\sum_{i \in S} a_i = B/2$ ?

Montrer par réduction du problème partition que la version décision du problème des transferts est NP-complète (étant donné une instance du problème des transferts et un entier $k$, est-il possible d'équilibrer les comptes avec au plus $k$ transferts ?).

Indication : étant donnée une instance de partition avec $n$ nombres, on pourra considérer une instance du problème des transferts avec $n+2$ personnes, $n$ étant déficitaires et 2 seulement bénéficiaire (et avec le même montant à recevoir).

**Question** 5 (1/10) — (en bonus, hors barême) Montrer que l'algorithme de la question 1 n'est pas $r$-approché, et ce quel que soit $r < 2$. On pourra s'inspirer de l'exemple introductif.