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

Année 2023–2024

Équipe enseignante :

Thomas Bellitto
Bruno Escoffier
Fanny Pascual

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) ?

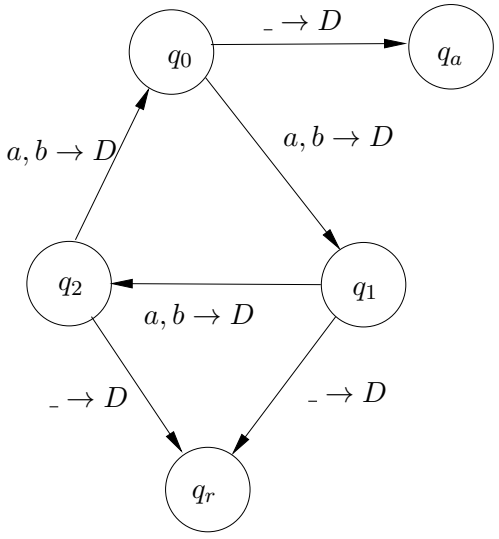


FIGURE 1 – Machine de Turing

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 ϕ
Question : ϕ est-elle satisfiable ?

Q 6.1 Soit n le nombre de variables de ϕ (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 ϕ 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 ϕ, 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 ϕ 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 ϕ (une formule booléenne 3FNC est une formule booléenne en forme normale conjonctive telle que chaque clause a 3 littéraux)
Question : ϕ 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 $\{x_1, \cdots, x_n\}$ (ℓ_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 ϕ 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 ϕ 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 ?

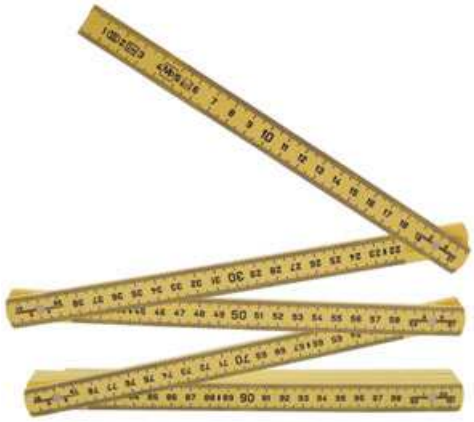


FIGURE 2 – Un mètre pliant ayant des sections de même longueurs.

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.

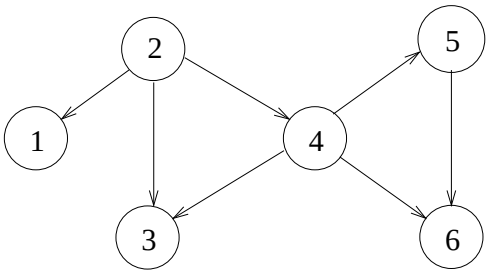


FIGURE 3 – Graphe G_1

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 σ 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.

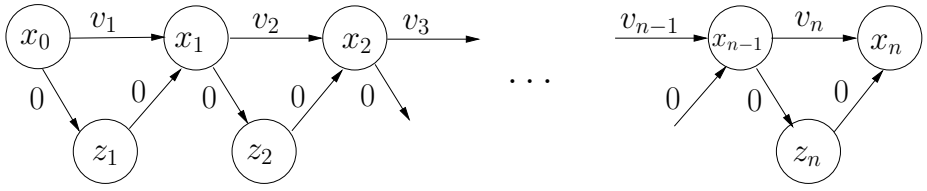


Figure 4 – Instance de CheminLongueurFixée correspondant à une instance $\{v_1, \ldots, v_n, B\}$ de SubsetSum.

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

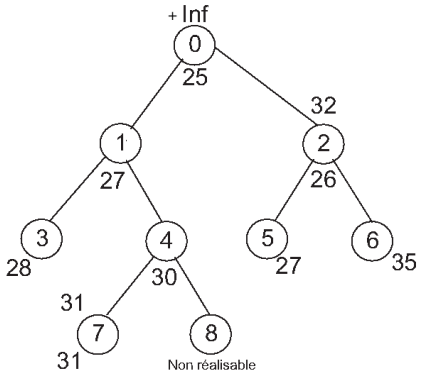
Q 17.1 Montrer que si P=NP, alors il existe un algorithme polynomial qui prend en entrée un graphe non orienté $G = (S, A)$ et qui retourne une plus grande clique de G.

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.

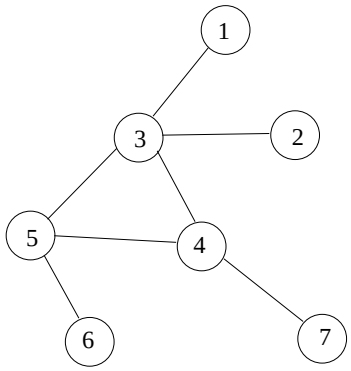


FIGURE 5 – Exemple

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 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.

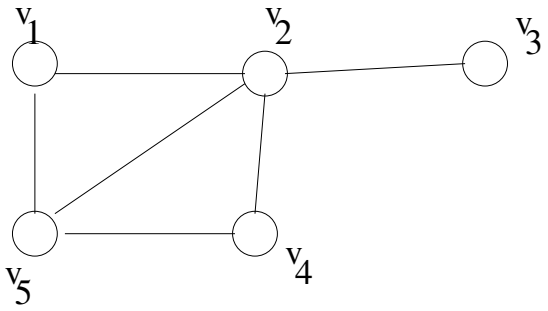
   On suppose connu le fait que ce problème est NP-complet.

   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_ℓ 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 σ_1 et σ_2, σ_1 consistant à mettre toutes les variables à vrai et σ_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 σ_1 et σ_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.

Couverture de graphe

1 Introduction

Soit un graphe $G = (V, E)$ non orienté, où V est l'ensemble des n sommets et E l'ensemble des m arêtes de G. Une couverture de G est un ensemble $V' \subseteq V$ tel que toute arête $e \in E$ a (au moins) une de ses extrémités dans V. Le but du problème Vertex Cover est de trouver une couverture contenant un nombre minimum de sommets.

Ce problème est NP-difficile (sa version décision [1] est NP-complète). Le but de ce projet est d'implémenter différents algorithmes, exacts et approchés, pour résoudre le problème Vertex Cover, et de les tester expérimentalement sur différentes instances.

Travail demandé

Chacune des sections suivantes comporte un certain nombre de questions. Il vous est demandé de :
- Réaliser un programme implantant les différents algorithmes et permettant de réaliser les tests demandés. Le code devra être commenté afin de permettre à un lecteur d'en comprendre le fonctionnement. Le choix du langage est libre.
- Ecrire un rapport (autour de 8 pages, 10 maximum) qui contient les réponses aux questions théoriques, les courbes/résultats numériques obtenus et leurs explications, interprétations,... Il est inutile d'inclure le code dans le rapport.

Le travail est à réaliser **en binôme**.

VOUS DEVEZ ETRE EN BINOME AVEC QUELQU'UN DE VOTRE GROUPE DE TD.

La date de rendu (rapport et code) sera donnée par votre chargé de TD. Tout retard sera pénalisé.

Le projet donnera lieu à une soutenance en salle machine. Lors de la soutenance, il pourra vous être demandé d'exécuter une ou plusieurs de vos fonctions sur un graphe fourni dans un fichier .txt dans un format identique au fichier "exempleinstance.txt" fourni sur le moodle. Vous devez donc écrire une méthode permettant de créer un graphe à partir d'un tel fichier texte.

2 Graphes

On pourra représenter un graphe par un tableau de listes d'adjacence. D'autres représentations sont également possibles. Il est également autorisé d'utiliser une bibliothèque de graphes pour gérer vos graphes (dans ce cas vous pouvez utiliser des méthodes existantes pour les opérations de base, mais pas pour la suite bien sûr).

2.1 Opérations de base

1. Coder une méthode prenant en paramètres un graphe G et un sommet v de G, et qui retourne un nouveau graphe G' obtenu à partir de G en supprimant le sommet v (et les arêtes incidentes). Précision : la méthode doit fonctionner même si les n sommets du graphe ne sont pas les entiers de 0 à $n-1$ (par exemple on peut vouloir supprimer le sommet 3 d'un graphe avec 4 sommets $\{0, 3, 6, 8\}$).
2. Généraliser la méthode précédente lorsqu'un ensemble de sommets est supprimé.

1. Etant donné un graphe G et un entier k, existe-t-il une couverture de taille au plus k ?

3. Coder une méthode prenant en entrée un graphe et renvoyant un tableau contenant les degrés des sommets du graphe, et une autre permettant de déterminer un sommet de degré maximum. Ici aussi, la méthode doit fonctionner même si les n sommets du graphe ne sont pas les entiers de 0 à $n-1$.

2.2 Génération d'instances

Afin de tester vos algorithmes, vous allez générer des graphes aléatoires.

1. Coder une méthode prenant en entrée un entier $n > 0$ et un paramètre $p \in]0,1[$, et qui renvoie un graphe sur n sommets, et où chaque arête (i,j) est présente avec probabilité p. Les tirages au sort seront indépendants pour les différentes arêtes.

3 Méthodes approchées

Méthode du couplage maximal

Un couplage est un ensemble d'arêtes n'ayant pas d'extrémité en commun.

```
algo_couplage(G)
----------------
Entrée: G=(V,E=(e_1,...,e_m)) un graphe
Sortie: Une couverture
----------------
  C = emptyset
  Pour i de 1 à m:
      Si aucune des deux extrémités de e_i n'est dans C, alors:
          Ajouter les deux extrémités de e_i à C
      Fin Si
  Fin Pour
  Renvoyer C
```

Algorithme glouton

```
algo_glouton(G)
---------------
Entrée: G=(V,E=(e_1,...,e_m)) un graphe
Sortie: Une couverture
----------------
  C = emptyset
  Tant que E n'est pas vide:
      Prendre un sommet v de degré maximum
          Ajouter v à C, et supprimer de E les arêtes couvertes par v
      Fin Si
  Fin Pour
  Renvoyer C
```

On rappelle que l'algorithme `algo-couplage` est 2-approché.

1. Montrer que l'algorithme glouton précédent n'est pas optimal. En déduire que l'algorithme glouton n'est pas r-approché, pour un certain r que à préciser.

2. Coder les deux méthodes précédentes. Les comparer (en fonction de n et p) du point de vue (1) de leur temps de calcul (2) de la qualité des solutions retournées.
3. (Facultatif) Montrer que l'algorithme glouton précédent n'est pas r-approché, quel que soit r.

Note sur les tests : Lorsque vous ferez des tests pour mesurer par exemple le temps de calcul sur des instances aléatoires, il vous est suggéré de :
- mesurer la taille N_{max} jusqu'à laquelle l'algorithme tourne rapidement, disons de l'ordre de quelques secondes.
- pour chaque taille d'instances $n \in \{N_{max}/10, 2N_{max}/10, 3N_{max}/10, \ldots, N_{max}\}$: générer quelques dizaines d'instances de taille n, mesurer le temps de calcul moyen t_n sur ces instances. Tracer la courbe t_n en fonction de n sur la base de ces 10 points. On pourra penser à utiliser des échelles logarithmiques si besoin.

NB : avant de faire des tests sur des instances générées, vérifiez toujours que votre code fonctionne sur quelques exemples simples !

4 Séparation et évaluation

4.1 Branchement

Le but de cette section est de faire un premier algorithme de branchement simple. Nous l'améliorerons dans les sections suivantes. Le branchement considéré ici est le suivant : prendre une arête $e = \{u, v\}$, et considérer deux cas : soit u est dans la couverture, soit v est dans la couverture.

Ainsi, partant d'un graphe initial G et d'un ensemble de sommets C vide représentant la solution, la première branche consiste à mettre u dans C et à raisonner sur le graphe où l'on a supprimé u (les arêtes incidentes à u sont couvertes) ; symétriquement, la deuxième consiste à mettre v dans C et à raisonner sur le graphe où l'on a supprimé v.

1. Coder cette méthode. Vous utiliserez une pile pour gérer les nœuds de l'arbre. Votre méthode devra renvoyer la meilleure solution. **Vérifier la validité de votre méthode sur des exemples simples** (vérifier que l'ensemble des noeuds visités est bien correct).
2. Tester cette méthode, en évaluant son temps de calcul en fonction de n. Vous pourrez faire varier la probabilité p de présence des arêtes. Commentez.

NB1 : il vous est demandé de faire des tests sur les temps de calcul. Il est également intéressant de tester le nombre de nœuds générés par les différentes méthodes. Cela est facultatif.

NB2 : dans les tests numériques, il est intéressant de regarder l'influence de p également (en testant plusieurs valeurs par exemple). On pourra aussi tester avec des valeurs comme $p = 1/\sqrt{n}$ pour avoir des graphes moins denses.

4.2 Ajout de bornes

Soit G un graphe, M un couplage de G et C une couverture de G. Alors

$$|C| \geq \max\{b_1, b_2, b_3\}$$

avec $b_1 = \lceil \frac{m}{\Delta} \rceil$ (où Δ est le degré maximum des sommets du graphe), $b_2 = |M|$, $b_3 = \frac{2n-1-\sqrt{(2n-1)^2-8m}}{2}$.

1. Montrer la validité de chacune des bornes b_1, b_2, b_3. Pour b_3, on pourra s'interroger sur le nombre maximal d'arêtes d'un graphe dont C est une couverture.

2. Insérer dans l'algorithme précédent (1) le calcul (en chaque nœud) d'une solution réalisable obtenue avec l'algorithme de couplage (2) le calcul d'une borne inférieure basée sur la relation précédente. Tester cette nouvelle méthode, et commenter.
3. (Facultatif) Evaluer l'intérêt d'utiliser aussi l'algorithme glouton pour générer des solutions réalisables. Evaluer l'efficacité des méthodes si l'on n'utilise que les bornes inférieures, ou que le calcul de solution réalisable (avec une borne inférieure triviale).

4.3 Amélioration du branchement

Lorsque l'on branche sur une arête $e = \{u, v\}$, dans la deuxième branche où l'on prend le sommet v dans la couverture, on peut supposer que l'on ne prend pas le sommet u (le cas où on le prend étant traité dans la première branche. Dans la 2ème branche, ne prenant pas u dans la couverture on doit alors prendre tous les voisins de u (et on peut les supprimer du graphe donc).

1. Modifier la méthode précédente en utilisant ce branchement amélioré. Tester la nouvelle méthode, et commenter les résultats obtenus.
2. Afin d'éliminer un maximum de sommets dans la deuxième branche, il semble intéressant de choisir le branchement de manière à ce que le sommet u soit de degré maximum dans le graphe restant. Tester la nouvelle méthode, et commenter les résultats obtenus.
3. (Facultatif) Dans un graphe G, si un sommet u est de degré 1, expliquer pourquoi il existe toujours une couverture optimale qui ne contient pas u. Insérer alors ce test permettant de supprimer un sommet sans brancher. Evaluer expérimentalement l'intérêt d'ajouter ce test.

4.4 Qualité des algorithmes approchés

1. Evaluer expérimentalement (en fonction de n) le rapport d'approximation des algorithmes de couplage et glouton. Donner également le pire rapport d'approximation obtenu lors des expérimentations, pour chacun des deux algorithmes.
2. (Facultatif) Proposer une ou plusieurs autres heuristiques, et les évaluer expérimentalement.