MINISTÈRES
ÉDUCATION
JEUNESSE
SPORTS
ENSEIGNEMENT
SUPÉRIEUR
RECHERCHE
Liberté
Égalité
Fraternité

EAE INF 2

SESSION 2026

AGREGATION
CONCOURS EXTERNE

Section : INFORMATIQUE

ÉTUDE D'UN PROBLÈME INFORMATIQUE

Durée : 6 heures

L'usage de tout ouvrage de référence, de tout dictionnaire et de tout matériel électronique (y compris la calculatrice) est rigoureusement interdit.

Il appartient au candidat de vérifier qu'il a reçu un sujet complet et correspondant à l'épreuve à laquelle il se présente.

Si vous repérez ce qui vous semble être une erreur d'énoncé, vous devez le signaler très lisiblement sur votre copie, en proposer la correction et poursuivre l'épreuve en conséquence. De même, si cela vous conduit à formuler une ou plusieurs hypothèses, vous devez la (ou les) mentionner explicitement.

NB : Conformément au principe d'anonymat, votre copie ne doit comporter aucun signe distinctif, tel que nom, signature, origine, etc. Si le travail qui vous est demandé consiste notamment en la rédaction d'un projet ou d'une note, vous devrez impérativement vous abstenir de la signer ou de l'identifier.
Le fait de rendre une copie blanche est éliminatoire.

Tournez la page S.V.P.

INFORMATION AUX CANDIDATS

Vous trouverez ci-après les codes nécessaires vous permettant de compléter les rubriques figurant en en-tête de votre copie.

Ces codes doivent être reportés sur chacune des copies que vous remettrez.

Concours	Section/option	Epreuve	Matière
EAE	6200A	102	9423

IMPRIMERIE NATIONALE – 26 0114 – D'après documents fournis

Coupes dans un graphe

Le problème de la coupe maximum est un problème de graphe qui trouve des applications en physique statistique, en intelligence artificielle, ou encore dans la conception de circuits imprimés.

Le sujet est découpé en six parties majoritairement indépendantes. La première introduit le concept de coupe maximum par des exemples. La deuxième propose une résolution naïve du problème, puis étudie le cas particulier des graphes bipartis. La troisième montre la NP-difficulté du problème. La quatrième présente un algorithme d'approximation avec une approche initialement probabiliste. La cinquième met en place un algorithme exact par séparation et évaluation. Enfin, la sixième et dernière partie présente le lien entre la résolution du problème de décision et du problème d'optimisation associés à la coupe maximum.

Consignes et notations

Notations. Si a et b sont des entiers, l'intervalle des entiers de a à b inclus est noté $[\![a, b]\!]$. Pour n un entier naturel, on note $\mathbb{N}_n$ l'ensemble des entiers de 0 à $n-1$, $\mathbb{N}_n = \{0, 1, \ldots, n-1\} = [\![0, n-1]\!]$.

La probabilité d'un événement e est notée $\mathbb{P}(e)$. L'espérance d'une variable aléatoire X est notée $\mathbb{E}(X)$.

Un *graphe* est un couple $G = (S, A)$ où S est un ensemble fini de sommets et $A \subseteq \mathcal{P}_2(S)$ un ensemble fini de paires de sommets représentant les arêtes. Dans ce sujet, tous les graphes sont non-orientés, non-pondérés, simple et sans boucle.

On identifiera une même grandeur écrite dans deux polices de caractères différentes, en italique du point de vue mathématique (par exemple n) et en police à chasse fixe du point de vue informatique (par exemple `n`).

Programmation. Les questions de programmation doivent être traitées en langage Python uniquement. On accordera une attention particulière à la présentation du code, notamment à la lisibilité des indentations.

Complexité. Sans précision supplémentaire, lorsqu'une question demande la complexité d'une fonction, il s'agira de la complexité temporelle dans le pire des cas. La complexité sera exprimée sous la forme $\mathcal{O}(f(n, m))$ où n et m sont les tailles des arguments de la fonction, et f une expression la plus simple possible. Dans l'ensemble du sujet, on supposera les opérations arithmétiques sur les entiers, sauf l'exponentiation, comme étant de complexité constante.

1 Définition d'une coupe et premiers exemples

Une *coupe* $C = (S_1, S_2)$ d'un graphe $G = (S, A)$ est une partition de l'ensemble S des sommets en deux parties éventuellement vides, c'est-à-dire que $S_1 \cup S_2 = S$, $S_1 \cap S_2 = \emptyset$. Une arête $a \in A$ est *coupée par* C si elle relie les deux parties de C, c'est-à-dire si elle est de la forme $a = \{s_1, s_2\}$ avec $s_1 \in S_1$ et $s_2 \in S_2$. La *taille* d'une coupe C, notée $\|C\|$, est le nombre d'arêtes coupées par C. Une coupe C d'un graphe G est *maximum* s'il n'existe pas de coupe C' de G telle que $\|C'\| > \|C\|$.

Exemple. La figure 1 représente le graphe G_0 et la coupe $C_0 = (S_1, S_2)$ où $S_1 = \{0, 2, 3\}$ et $S_2 = \{1, 4\}$. Les arêtes coupées par C_0 sont représentées en pointillés. La coupe C_0 est donc de taille $\|C_0\| = 4$.

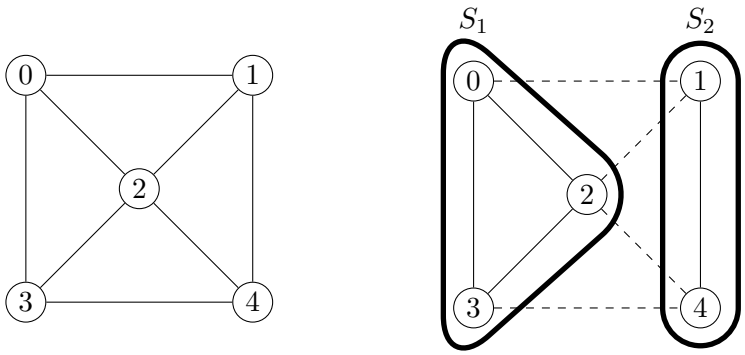


FIGURE 1 – Le graphe G_0 (à gauche) et la coupe C_0 du graphe G_0 (à droite).

Question 1 Soient les coupes $C_1 = (\{2\}, \{0, 1, 3, 4\})$ et $C_2 = (\{2, 3\}, \{0, 1, 4\})$ du graphe G_0. Ces coupes sont-elles maximums ?

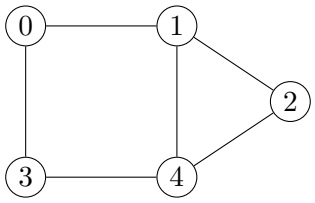


FIGURE 2 – Le graphe G_1.

Question 2 Donner une coupe maximum pour le graphe G_1 représenté sur la figure 2. Justifier qu'elle est bien maximum.

2 Premières approches

2.1 Approche naïve

Dans cette section, on calcule la coupe maximum d'un graphe de façon naïve en parcourant toutes les coupes possibles et en en retenant une de taille maximum au fur et à mesure. On commence par des questions préliminaires, puis on verra comment implémenter cette approche naïve.

Question 3 Dans un graphe à n sommets, combien existe-t-il de coupes ? On considère que si $C = (S_1, S_2)$ est une coupe d'un tel graphe, alors (S_2, S_1) est une coupe différente de C.

On considère que les sommets des graphes sont des entiers consécutifs à partir de 0 et on représente ces graphes par des listes de listes d'adjacence. Pour pouvoir manipuler des coupes en Python, dans l'ensemble du sujet, on les représente par des listes de booléens. Si une coupe $C = (S_1, S_2)$ du graphe G est représentée par la liste `C`, alors pour tout sommet `i`, `C[i]` vaut `True` si et seulement si `i` appartient à S_1. Par exemple, le graphe G_0 est représenté par la liste `[[1, 2, 3], [0, 2, 4], [0, 1, 3, 4], [0, 2, 4], [1, 2, 3]]` et la coupe C_0 est représentée par la liste `[True, False, True, True, False]`.

Question 4 Écrire une fonction `cardinal_S1(C)` qui prend en argument une coupe `C` d'un graphe et qui renvoie le nombre de sommets appartenant à S_1 dans `C`.

Question 5 Écrire une fonction `taille(G, C)` qui prend en arguments un graphe `G` et une coupe `C` de ce graphe et qui renvoie la taille $\|C\|$ de `C`.

À présent, on souhaite implémenter l'approche naïve décrite précédemment. Pour cela, on suppose dans un premier temps que l'on dispose d'une fonction `suivante(C)` qui prend en argument une coupe `C` et qui ne renvoie rien, mais modifie la coupe pour passer à la coupe suivante de manière à ce qu'en appliquant cette fonction suffisamment de fois à la coupe `[False] * n`, on parcourt toutes les coupes d'un graphe de `n` sommets en ne passant qu'une seule fois par chaque coupe.

On suppose qu'un appel à `suivante(C)` lorsque `C` est la dernière coupe parcourue ne crée pas d'erreur et modifie `C` arbitrairement sans rien renvoyer.

Question 6 Écrire une fonction `coupe_max_naive(G)` qui prend en argument un graphe `G` et qui renvoie une coupe maximum en utilisant la fonction `suivante`.

Maintenant que l'on a implémenté la fonction principale, on cherche à implémenter la fonction `suivante`. On propose de commencer par une version utilisant des fonctions de conversions coupes / entiers. On verra ensuite comment implémenter une seconde version permettant de calculer directement la coupe suivante sans passer par les entiers.

Question 7 Écrire une fonction `entier_vers_coupe(e, n)` qui prend en arguments un entier `e` et un nombre de sommets `n` et renvoie la coupe associée, c'est-à-dire une liste de booléens `res` de longueur `n` telle que $\sum_{i=0}^{n-1} \texttt{res[i]} \times 2^i = \texttt{e}$. On suppose que $e \leqslant 2^n - 1$ et on attend une complexité linéaire en n.

Question 8 Écrire une fonction `coupe_vers_entier(C)` qui prend en argument une coupe `C` et qui renvoie l'entier `e` associé à cette coupe, c'est-à-dire tel que `entier_vers_coupe(e, len(C))` est égal à `C`. On attend une complexité linéaire en $n = \texttt{len(C)}$.

Question 9 En déduire une implémentation de la fonction `suivante`.

Question 10 Donner, en justifiant, la complexité de `coupe_max_naive` qui utilise la fonction `suivante` implémentée dans la question précédente.

En s'inspirant de l'algorithme d'incrémentation des entiers en base 2, il est possible de passer d'une coupe à la suivante en ne parcourant que les cases à modifier.

Question 11 Écrire une fonction `suivante2(C)` qui prend en argument une coupe `C` et qui a le même effet que `suivante(C)` mais en ne parcourant que les cases à modifier dans `C`.

Question 12 Donner, en justifiant, la complexité amortie de `suivante2` si on l'utilise pour parcourir toutes les coupes possibles.

Question 13 Quelle est la complexité de `coupe_max_naive` en utilisant la fonction `suivante2` à la place de `suivante` ?

2.2 Graphes bipartis

Dans le cas général, le problème de la coupe maximum est difficile. Dans cette section, on s'intéresse au cas particulier des graphes bipartis pour lesquels ce problème admet une résolution efficace.

Question 14 Écrire une fonction `est_biparti(G)` qui prend en argument un graphe `G` et qui renvoie :

- `(False, ([], []))` si `G` n'est pas biparti
- `(True, (X, Y))` où `X` et `Y` sont des listes de sommets de `G` telles que chaque sommet est dans exactement l'une des deux listes et qu'aucune arête de `G` ne relie deux sommets d'une même liste, sinon.

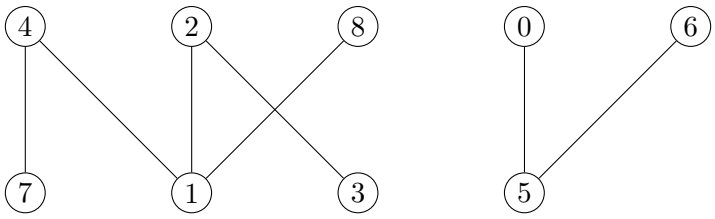


FIGURE 3 – Un graphe biparti G_2.

Question 15 Donner une coupe maximum du graphe biparti G_2 représenté en figure 3. Justifier la maximalité de la coupe.

Question 16 Décrire brièvement un algorithme efficace permettant de calculer une coupe maximum dans un graphe biparti quelconque.

Question 17 Écrire une fonction `coupe_max_biparti(G)` qui prend en argument un graphe biparti et qui renvoie une coupe maximum. On attend une complexité linéaire en $|S| + |A|$ pour un graphe $G = (S, A)$, mais on ne demande pas de justifier cette complexité.

On s'intéresse pour terminer cette section au cas où un graphe biparti admet une unique coupe maximum, c'est-à-dire au cas où les deux seules coupes maximum d'un graphe sont (S_1, S_2) et (S_2, S_1), pour un même S_1 et un même S_2 : la coupe maximum est unique à interversion de S_1 et S_2 près.

Question 18 Donner un graphe biparti pour lequel il n'y a pas unicité de la coupe maximum.

Question 19 Est-ce qu'un graphe biparti connexe admet nécessairement une unique coupe maximum ? Justifier.

Question 20 Est-ce qu'un graphe biparti qui admet une unique coupe maximum est nécessairement connexe ? Justifier.

3 NP-complétude

Dans cette partie, on souhaite montrer que le problème de la coupe maximum est un problème complexe. On s'intéresse ici à une version décisionnelle, `COUPE MAX` :

- **instance** : un graphe non orienté $G = (S, A)$ et un entier $k \in \mathbb{N}$.
- **question** : est-ce que G possède une coupe C telle que $\|C\| \geqslant k$?

On souhaite montrer que `COUPE MAX` est NP-complet par une réduction depuis le problème `3SAT` :

- **instance** : une formule propositionnelle φ en 3-FNC, c'est-à-dire en forme normale conjonctive avec exactement trois littéraux par clause.
- **question** : est-ce que φ est satisfiable ?

dont on admet qu'il est NP-complet.

3.1 NP-complétude de `STABLE`

Avant de se ramener au problème `COUPE MAX`, on montre la NP-complétude d'un autre problème de décision sur les graphes qui s'intéresse à la recherche d'un stable dans un graphe.

Étant donné un graphe non orienté $G = (S, A)$, un *stable* de G est un ensemble $X \subseteq S$ tel que deux sommets de X ne sont jamais adjacents, c'est-à-dire $\forall s, t \in X, \{s, t\} \notin A$.

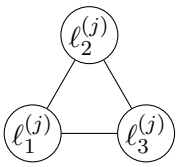
Le problème `STABLE` est le suivant :

- **instance** : un graphe non orienté $G = (S, A)$ et un entier $k \in \mathbb{N}$.
- **question** : est-ce que G possède un stable X tel que $|X| \geqslant k$?

Question 21 Montrer que `STABLE` $\in$ NP.

Soit $\mathcal{V} = \{x_1, x_2, \ldots, x_n\}$ un ensemble de variables et $\varphi = \bigwedge_{j=1}^{m} C_j$ une formule en 3-FNC sur $\mathcal{V}$, où chaque C_j est une clause avec exactement trois littéraux. On construit le graphe $G_\varphi = (S_\varphi, A_\varphi)$ de la manière suivante :

- pour chaque clause $C_j = \ell_1 \vee \ell_2 \vee \ell_3$, où les ℓ_i sont des littéraux, on construit un sous-graphe avec trois sommets $\ell_1^{(j)}$, $\ell_2^{(j)}$ et $\ell_3^{(j)}$ adjacents deux à deux :



- pour chaque paire $x_i^{(j)}$, $\neg x_i^{(j')}$ d'une variable et de sa négation apparaissant dans deux sous-graphes distincts, on rajoute une arête entre le sommet $x_i^{(j)}$ et le sommet $\neg x_i^{(j')}$ correspondant.

Exemple. la figure 4 représente le graphe G_{φ_0} pour φ_0 la formule définie par :

$$\varphi_0 = (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

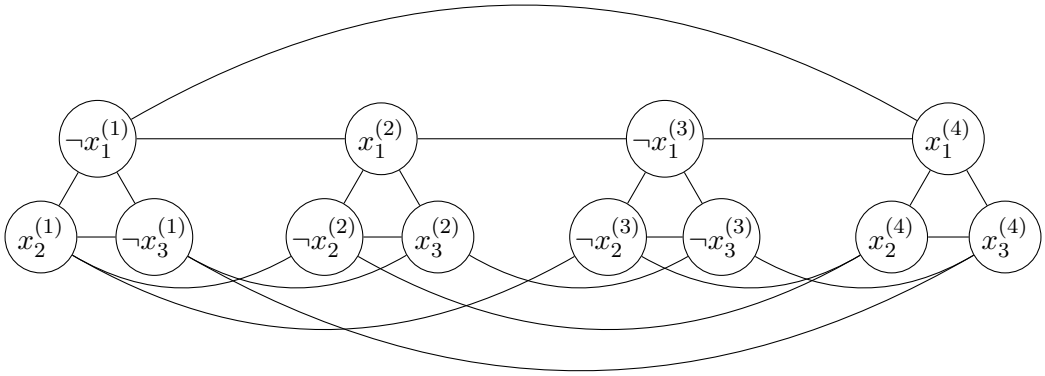


FIGURE 4 – Le graphe G_{φ_0}.

Question 22 Déterminer un modèle de φ_0, où φ_0 est la formule définie dans l'exemple précédent. À partir de ce modèle, déterminer un stable de cardinal 4 dans le graphe G_{φ_0} de la figure 4.

On considère $\varphi = \bigwedge_{j=1}^{m} C_j$ une formule propositionnelle quelconque en 3-FNC.

Question 23 Montrer que φ est satisfiable si et seulement si G_φ possède un stable de cardinal m.

Question 24 En déduire que `STABLE` est NP-complet.

3.2 De `STABLE` à `COUPE MAX`

On cherche maintenant à réduire le problème `STABLE` au problème `COUPE MAX`. On considère un graphe $G = (S, A)$. À partir de G, on définit le graphe $G' = (S', A')$ de la manière suivante :

- S' contient tous les sommets de S, ainsi qu'un nouveau sommet w, et, pour chaque arête $a \in A$, deux nouveaux sommets u_a et v_a, c'est-à-dire :
$$S' = S \cup \bigcup_{a \in A} \{u_a, v_a\} \cup \{w\}$$
- tous les sommets de $S' \setminus \{w\}$ sont adjacents à w ; de plus, pour chaque arête $a = \{s, t\} \in A$, A' contient les arêtes $\{s, u_a\}$, $\{u_a, v_a\}$ et $\{v_a, t\}$, c'est-à-dire :
$$A' = \{\{x, w\} \mid x \in S' \setminus \{w\}\} \cup \bigcup_{a=\{s,t\} \in A} \{\{s, u_a\}, \{u_a, v_a\}, \{v_a, t\}\}$$

Tournez la page S.V.P.

Exemple. La figure 5 montre la construction d'un « gadget » à partir d'une arête de G.

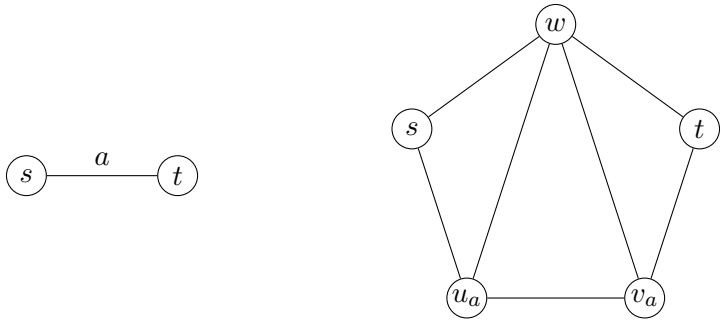


FIGURE 5 – Transformation d'une arête $a = \{s, t\}$ dans G en un gadget dans G'.

Question 25 Montrer que s'il existe un stable X de cardinal k dans G, il existe une coupe $C = (S_1, S_2)$ de cardinal $k + 4|A|$ dans G', telle que $X \subseteq S_1$ et $w \in S_2$.

On cherche à montrer la réciproque. Pour les deux questions suivantes, on considère $C = (S_1, S_2)$ une coupe de cardinal $k + 4|A|$ dans G' et on veut montrer qu'il existe un stable X de cardinal k dans G.

Quitte à échanger les rôles, on peut supposer que $w \in S_2$. On pose $Y = S \cap S_1$, c'est-à-dire l'ensemble des sommets de S_1 qui ne sont pas de la forme u_a ou v_a pour une certaine arête a.

On pose enfin $A_Y = \mathcal{P}_2(Y) \cap A$, c'est-à-dire l'ensemble des arêtes de G (pas de G') qui sont entre deux sommets de Y.

Question 26 Montrer que $\|C\| \leqslant |Y| + 3|A_Y| + 4(|A| - |A_Y|)$.

Question 27 En déduire qu'il existe un stable $X \subseteq Y$ de G tel que $|X| = k$.

Question 28 En déduire que `COUPE MAX` est NP-complet.

4 Algorithme d'approximation

4.1 Approximation probabiliste

Soit G un graphe non orienté. On suppose que G admet une coupe maximum C^* de taille k^*. On cherche à mettre en place un algorithme probabiliste qui renvoie une coupe C de taille k vérifiant $k \geqslant \frac{k^*}{2}$ avec grande probabilité.

On considère l'algorithme suivant : chaque sommet est placé aléatoirement dans S_1 ou S_2 (avec probabilité $\frac{1}{2}$ d'être dans S_1), et les tirages pour les différents sommets sont indépendants.

On suppose qu'on a importé le module `random` avec la commande suivante :

```
import random
```

La fonction `randrange` du module `random` prend en arguments deux entiers a et b et renvoie un entier tiré aléatoirement et uniformément entre a inclus et b exclu.

Question 29 Écrire une fonction `coupe_proba(G)` qui prend en argument un graphe G et renvoie une coupe (S_1, S_2) obtenue avec l'algorithme précédent.

Question 30 Soit $C = (S_1, S_2)$ une coupe renvoyée par l'algorithme. Montrer que $\|C\|$ a une espérance égale à $\frac{|A|}{2}$.

Question 31 En utilisant la fonction `coupe_proba`, écrire une fonction `coupe_las_vegas(G)`, correspondant à un algorithme de type Las Vegas, qui prend en argument un graphe G et renvoie une coupe de taille supérieure ou égale à $\frac{|A|}{2}$.

Pour $k \in \mathbb{N}^*$, si X est une variable aléatoire entière à valeurs dans $[\![0, k]\!]$ et d'espérance $\frac{k}{2}$, on admet l'inégalité suivante :

$$\mathbb{P}\left(X \geqslant \frac{k}{2}\right) \geqslant \frac{1}{k}$$

Question 32 Montrer que l'espérance du temps de calcul de la fonction `coupe_las_vegas`, quitte à la modifier, est polynomiale en $|S|$ et $|A|$.

4.2 Dérandomisation

On cherche à dérandomiser l'algorithme précédent, c'est-à-dire à le transformer en algorithme n'utilisant pas d'aléatoire, tout en gardant les mêmes garanties d'approximation. L'idée est de parcourir l'ensemble des sommets de S et de les attribuer à S_1 ou S_2 selon une heuristique à déterminer.

Tournez la page S.V.P.

Le schéma général de l'algorithme est le suivant :

```
   Entrée : Graphe G = (ℕ_n, A)
1  Début algorithme
2  |  Poser S_1 = ∅ et S_2 = ∅.
3  |  Pour i de 0 à n − 1 Faire
4  |  |  Choisir de rajouter le sommet i à S_1 ou à S_2.
5  |  Renvoyer (S_1, S_2)
```

L'étape clé est évidemment le choix d'attribution d'un sommet à S_1 ou à S_2 à la ligne 4 de l'algorithme. Pour mieux comprendre les différents états au cours de l'algorithme, pour $i \leqslant n$, on note $S_{i,1}$ et $S_{i,2}$ les ensembles de sommets S_1 et S_2 après avoir fait le choix pour le sommet $i-1$.

On suppose que l'algorithme a été exécuté jusqu'au choix du sommet $i-1$, avec $i < n$. On note $X_i = \mathbb{N}_n \setminus \mathbb{N}_i$ l'ensemble des sommets pour lesquels aucun choix n'a été fait. On note également A_i l'ensemble des arêtes coupées par $(S_{i,1}, S_{i,2})$ à l'étape i, et B_i l'ensemble des arêtes ayant au moins une extrémité dans X_i.

On envisage dans un premier temps de faire les choix restants de manière aléatoire et uniforme (chaque sommet a une chance sur 2 d'être attribué à S_1 et à S_2). On note c_i le nombre moyen d'arêtes coupées par la coupe obtenue à partir de $(S_{i,1}, S_{i,2})$ en complétant les attributions aléatoirement.

Question 33 Montrer que $c_i = |A_i| + \dfrac{|B_i|}{2}$.

On veut remplacer les choix aléatoires par des choix déterministes qui garantissent que $c_0 \leqslant c_1 \leqslant \cdots \leqslant c_n$. Pour ce faire, on note $x_{i,1}$ le nombre de voisins du sommet i dans $S_{i,1}$ et $x_{i,2}$ le nombre de voisins du sommet i dans $S_{i,2}$.

Question 34 Montrer que $c_{i+1} - c_i = \begin{cases} \frac{1}{2}(x_{i,1} - x_{i,2}) & \text{si le sommet } i \text{ est ajouté à } S_2 \\ \frac{1}{2}(x_{i,2} - x_{i,1}) & \text{sinon.} \end{cases}$

Question 35 En déduire une manière de faire chaque choix pour obtenir un algorithme déterministe qui est une $\frac{1}{2}$-approximation du problème de la coupe maximum.

Question 36 Écrire une fonction `coupe_approx(G)` qui prend en argument un graphe G et renvoie une coupe selon l'algorithme déterministe décrit à la question précédente.

Question 37 Déterminer la complexité temporelle de la fonction `coupe_approx` en fonction de $|S|$ et $|A|$.

5 Algorithme optimal par séparation et évaluation

On souhaite écrire un algorithme de séparation et évaluation (*branch and bound*) pour calculer efficacement une coupe maximum. On s'intéresse dans un premier temps à un algorithme de retour sur trace (*backtracking*), où on construit une solution en complétant des solutions partielles, sur le même schéma que l'algorithme d'approximation.

Pour un graphe $G = (S, A)$ avec $S = \mathbb{N}_n$, on suppose qu'une solution partielle au problème est une liste de booléens `C` de taille i strictement inférieure à n. L'interprétation d'une telle liste est une coupe partielle $C_i = (S_{i,1}, S_{i,2})$ construite de la manière suivante :

- pour $s \in \mathbb{N}_i$, `C[s]` vaut `True`, alors $s \in S_{i,1}$;
- pour $s \in \mathbb{N}_i$, `C[s]` vaut `False`, alors $s \in S_{i,2}$;
- pour $s \geqslant i$, il n'a pas encore été déterminé si $s \in S_{i,1}$ ou $s \in S_{i,2}$.

Question 38 Écrire une fonction `retour_sur_trace(G)` qui calcule une coupe maximum d'un graphe G donné en argument en utilisant un algorithme de type retour sur trace en complétant des solutions partielles au format décrit précédemment.

Question 39 Comparer la complexité temporelle de la fonction `retour_sur_trace` et de la fonction `coupe_max_naive`.

Pour une coupe partielle $C_i = (S_{i,1}, S_{i,2})$, avec $S_{i,1} \cup S_{i,2} \subseteq \mathbb{N}_i$, et pour $X_i = \mathbb{N}_n \setminus \mathbb{N}_i$, on définit les ensembles d'arêtes suivants :

- A_i est l'ensemble des arêtes entre un sommet de $S_{i,1}$ et un sommet de $S_{i,2}$, c'est-à-dire coupées par C_i (comme dans la partie précédente) ;
- B'_i est l'ensemble des arêtes entre **deux** sommets de X_i (à différencier de B_i de la partie précédente) ;
- pour $s \in X_i$, $B_{i,1}(s)$ (resp. $B_{i,2}(s)$) est l'ensemble des arêtes entre s et un sommet de $S_{i,1}$ (resp. $S_{i,2}$).

On définit une heuristique d'évaluation $h(C_i)$ par :

$$h(C_i) = |A_i| + |B'_i| + \sum_{s \in X_i} \max(|B_{i,1}(s)|, |B_{i,2}(s)|)$$

Question 40 Montrer que l'heuristique h est une heuristique admissible, c'est-à-dire que pour une coupe partielle $C_i = (S_{i,1}, S_{i,2})$, $h(C_i)$ est supérieur ou égale à la taille maximale d'une coupe $C = (S_1, S_2)$ telle que $S_{i,1} \subseteq S_1$ et $S_{i,2} \subseteq S_2$.

Question 41 Écrire une fonction `separation_evaluation(G)` qui calcule une coupe maximum d'un graphe G donné en argument en utilisant un algorithme de type séparation et évaluation. Détailler l'heuristique de séparation utilisée, c'est-à-dire l'ordre dans lequel les solutions partielles sont explorées.

6 Utilisation d'un oracle

Dans cette partie, on veut savoir s'il est possible de reconstruire une coupe maximum si on sait résoudre le problème de décision `COUPE MAX`. On suppose disposer d'une fonction efficace `oracle(G, k)` qui résout le problème de décision `COUPE MAX`, c'est-à-dire qui prend en arguments un graphe $G = (S, A)$ et un entier $k \in \mathbb{N}$ et renvoie `True` s'il existe une coupe de G de taille supérieure ou égale à k et `False` sinon.

Question 42 Écrire une fonction `taille_coupe_max(G)` qui prend en argument un graphe $G = (S, A)$ et renvoie la taille maximale d'une coupe de G. En supposant qu'un appel à `oracle` est en temps constant, la fonction doit avoir une complexité logarithmique en $|S| + |A|$ et on demande de le justifier.

Question 43 Décrire en français un algorithme qui prend en argument un graphe $G = (S, A)$ et renvoie une coupe maximum de G. En supposant qu'un appel à `oracle` est en temps constant, l'algorithme doit avoir une complexité polynomiale en $|S| + |A|$. Justifier la correction de l'algorithme.

Question 44 Écrire une fonction `reconstruire_coupe_max(G)` qui implémente cet algorithme.

* *
*