Master Informatique Fondamentale (1ère année)
Algorithmes d'approximation — **TD1**

Empaquetage et arbres de Steiner

1er octobre 2004

Exercice 1 : Notions de base du cours.

Question 1. *Donner une définition précise de « problème de maximisation » et « α-approximation » (α constant) et « $\alpha(n)$-approximation » (le facteur d'approximation dépend de la taille de l'instance) pour un tel problème.*

Réponse : Soit P un problème donné et S l'ensemble de ses solutions pour une instance I. Soit $\mathbb{S}$ un ensemble contenant toutes les solutions de P pour toutes ses instances. Soit $\rho : \mathbb{S} \to \mathbb{R}$, on peut voir ρ comme un critère d'évaluation pour les solutions. Un problème de maximisation est donc de trouver : $m \in S, \forall s \in S, \rho(s) \leq \rho(m)$. On dit que m est une solution maximale, ou optimale, de P pour son instance I et on note $OPT(I) = \rho(m)$.

Une $\alpha(n) - approximation$ d'un problème de maximisation est un algorithme qui s'exécute en temps polynomial A et tel que pour toute instance I de P on ait : $\rho(A(I)) \geq \alpha(|I|)OPT(I)$. Une $\alpha - approximation$ est une $\alpha(n) - approximation$ telle que : $\forall n \in (N), \alpha(n) = \alpha$. □

Question 2. *Donner une définition précise de « instance critique » pour un algorithme approximant un problème de maximisation.*

Réponse : On dit que $(I_n)_{n\in\mathbb{N}}$ est une famille d'instances critiques pour un algorithme d'approximation A si et seulement si :

$$\liminf_{n\in\mathbb{N}} \frac{\rho(A(I_n))}{OPT(I_n)\alpha(|I_n|)} = 1.$$

Moralement, cette famille est les pires cas de figure pour un algorithme d'approximation. □

Exercice 2 : De l'algorithme élaboré collectivement en cours. Dans un arbre, on appelle *fourche* une chaine contenant deux sommets de degré 1 et contenant au plus un sommet de degré supérieur ou égal à 3. En cours nous avons esquissé un algorithme qui consiste à former un cycle en reliant les deux extrémités d'une fourche, à contracter ce cycle puis à itérer. Dans une deuxième phase, les cycles obtenus sont fusionnés pour obtenir un tour. Nous ne nous intéressons ici qu'à la première phase de l'algorithme.

Question 3. *Montrer que tout arbre contient au moins une fourche.*

Réponse : On raisonne par induction : un arbre à deux sommets est une fourche et donc en contient une. Soit T un arbre à n sommets, soit b une de ses feuilles. Par induction $T \setminus \{b\}$ contient une fourche que l'on notera $F = (a_0, \ldots, a_m)$. Soit c le voisin de b dans T, si c n'est pas dans la fourche, on a une fouche pour T. Si $c = a_i$: On note $d_1(e)$ le degré d'un sommet e dans T et $d_2(e)$ son degré dans $T \setminus \{b\}$. On distingue trois cas :

- $d_2(a_i) \geq 3 \Rightarrow d_1(a_i) = d_2(a_i) + 1 \geq 3$. Ainsi, F est une fourche dans T.
- $d_2(a_i) = 2$: S'il n'y a pas de sommet dans F de degré supérieur à 3 dans $T\setminus\{b\}$, F est une fourche dans T. Sinon, soit a_k ce sommet. Si $k > i$, $(a_0, \ldots, a_i, b)$ est une fourche dans $T \setminus \{b\}$; sinon, $(b, a_i, \ldots, a_m)$ en est une.
- $d_2(a_i) < 2 \Rightarrow d_1(a_i) < 3$. Donc, F est une fourche dans $T \setminus \{b\}$.

□

Question 4. *Donner une structure de donnée efficace pour trouver une fourche en temps polynômial, la contracter, puis itérer ce processus jusqu'à épuisement.*

Réponse : L'idée est de trouver un sommet de degré supérieur à 3 de profondeur maximale et d'extraire la fourche ne contenant que des sommets de profondeur supérieure.

L'algorithme :

- Prendre un sommet au hasard et marquer la profondeur des sommets et leur degré en effectuant un parcours en largeur à partir de ce sommet.
- Trier les sommets par ordre décroissant de profondeur.
- Prendre le 1^{er} sommet de degré supérieur à 3.
- Prendre 2 voisins de profondeur plus grande et descendre jusqu'aux feuilles.
- Enlever la chaine trouvée et mettre à jour les degrés. (la profondeur ne change jamais).
- Recommencer jusqu'à épuisement.

□

Exercice 3 : Empaquetage. Le problème de l'empaquetage (unidimensionnel), ou *bin packing*, est le suivant : étant donné une séquence d'objets (unidimensionels) $(a_1, a_2, \ldots, a_n)$ de taille $t(a_i) = t_i \in]0; 1]$ (pour $1 \leq i \leq n$), ranger ces objets dans des boites de taille unitaire en minimisant le nombre de boites utilisées. Ce problème est NP-difficile ainsi que de nombreuses généralisations. Nous cherchons donc à l'approximer.

Dans cet exercice, nous nous intéressons uniquement aux algorithmes en ligne (*on-line*), c'est à dire des algorithmes qui considèrent chaque objet l'un après l'autre (dans l'ordre où ils sont donnés) et qui le range une fois pour toutes avant de passer aux suivants. L'algorithme suivant (*next fit*) est un algorithme en ligne :

1. prendre une boite vide et l'ouvrir ;
2. considérer un nouvel objet O tant qu'il en reste ;
3. si O rentre dans la boite ouverte le mettre dedans et aller en 2 ;
4. sinon fermer la boite, en ouvrir une nouvelle, y mettre O et aller en 2 ;

Question 5. *Trouver des instances aussi mauvaises que vous pouvez pour l'algorithme* next fit.

Réponse : On considère les objets $(O_n)_{0 \leq n \leq 2N}$ tels que :

- $t(O_{2p}) = 1$
- $t(O_{2p+1}) = \epsilon$

Next Fit ne mettra qu'un objet par boite, alors que si ϵ est choisi suffisament petit, on peut se contenter de mettre les objets pairs dans une boite chacun, et tous les impairs dans la même boite, ce qui ne prendra que $N + 1$ boites. □

Question 6. *Montrer que* next fit *est une α-approximation du problème de l'empaquetage pour une constante α à déterminer. En revenant si nécessaire sur la question 5 obtenir une analyse exacte de l'algorithme.*

Réponse : Next Fit est une $2 - approximation$. En effet, on a trivialement : $\sum_{i=0}^{n} t_i \leq OPT(I)$. On considère alors deux boites consécutives quelconques remplies à r_1 et r_2. On a : $r_1 + r_2 > 1$ sinon Next Fit aurait mis au moins le premier objet de la deuxième boite dans la première.

Ainsi, on a : $NextFit(I) \leq 2 \sum_{i=0}^{n} t_i \leq 2OPT(I)$, ce qui prouve le résultat. □

Question 7. *Montrer qu'aucun algorithme en ligne ne peut faire mieux qu'une $\frac{4}{3}$-approximation. On pourra considérer des instances du type :*

- *on fixe* $0 < \epsilon < \frac{1}{3}$;
- $t_i = \begin{cases} \frac{1}{3} + \epsilon & \textit{pour } 1 \leq i \leq p \\ \frac{2}{3} - \epsilon & \textit{pour } p < i \leq n \end{cases}$

p reste à déterminer judicieusement en fonction de l'algorithme considéré ; pour cela, on procédera à une discussion sur les nombres x et y de boites contenant 2 et 1 objet(s) (respectivement) après le traitement par l'algorithme des p premiers objets.

Réponse : Prenons, comme conseillé, pour $0 < \epsilon < \frac{1}{3}$, des objets de taille t_i. La solution optimale est alors

$$OPT(n) = \begin{cases} \frac{n}{2} \text{ si } p > n - p \\ n - p \text{ sinon.} \end{cases}$$

Considérons maintenant l'algorithme en ligne. Au bout de p objets, on n'a eu que des objets de taille $\frac{1}{3} + \epsilon$, on compte alors x le nombre de boîtes contenant 2 objets, $y = p - 2x$ le nombre de boîtes contenant 1 objet. Les objets suivants, plus grands, pourront remplir au plus y des boîtes déjà présentes.

- Si $x > \frac{p}{3}$, on pose $n = 2p$: $x > \frac{p}{3} \geq \frac{n-p}{3}$. Donc $Algo(n) \geq x + n - p \geq \frac{4}{3}(n - p) = \frac{4}{3}OPT(n)$.
- Si $x \leq \frac{p}{3}$, $y \geq \frac{p}{3}$. Finalement, on décide de s'arrêter là, et prendre $n = p$. Alors $Algo(n) = x + y = \frac{p-y}{2} + y \geq \frac{4}{6}p \geq \frac{4}{3}\frac{n}{2} = \frac{4}{3}OPT(n)$.

Il est donc toujours possible de tromper l'algorithme pour qu'il reste une $\frac{4}{3}$-approximation. □

Exercice 4 : Arbres de Steiner. Le problème de l'arbre de Steiner est le suivant : étant donné un graphe G non orienté, muni d'une fonction de coût positive sur les arêtes, et dont certains sommets sont étiquetés **Requis**, trouver un arbre de coût minimum dans G contenant tous les sommets étiquetés **Requis**. Ce problème est NP-difficile et nous cherchons à l'approximer.

Question 8. *Montrer que d'un algorithme d'approximation du problème de l'arbre Steiner métrique (graphe complet et fonction de poids vérifiant l'inégalité triangulaire) on déduit un algorithme d'approximation du problème de l'arbre de Steiner général avec le même facteur d'approximation. (On s'appuiera sur la notion de réduction* isofacteur *de* Π *vers* Π' *: une instance* I *de* Π *se réduit en temps polynômial en une instance* I' *de* Π' *puis une solution approchée de* I *se construit en temps polynômial à partir d'une solution approchée de* I' *avec conservation du facteur d'approximation).*

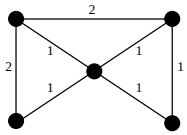
Réponse : Soit $\mathfrak{A}$ une α-approximation du problème de Steiner métrique, et $I = (G, w)$ une instance du problème général. A partir de la fonction de poids w, on calcule w', poids minimal d'un chemin entre deux points, de façon polynomiale (par exemple par algorithme de Dijkstra, w étant positif). La fonction w' vérifie l'inégalité triangulaire, on peut lui appliquer l'algorithme $\mathfrak{A}$. On obtient un arbre qui minimise (à α près) les poids des chemins entre les points.

Pour revenir au problème initial, il suffit de retracer les chemins minimaux qui nous intéressent entre les points. En faisant ainsi, on ne peut qu'abaisser le poids total, puisque quand des chemins se superposent, on ne compte le poids qu'une fois. □

Nous nous intéressons donc maintenant uniquement à la version *métrique* du problème.

Question 9. *Dans une instance de Steiner métrique, que peut-on dire de l'arbre couvrant de poids minimum du sous-graphe induit par l'enemble* R *des sommets étiquetés* ***Requis*** *? Justifier votre réponse.*

Réponse : Il faut d'abord qu'un tel sous-graphe soit complet. Mais même dans ce cas, on voit que le sous-arbre correspondant n'a pas forcément grand-chose à voir avec le résultat final.



Ici, par exemple, si $a, b, c, d \in R$, l'arbre a pour poids 5, alors qu'un arbre minimal (l'étoile de centre e) a pour poids 4. Et les arêtes des deux peuvent être complètement disjointes.

Par contre, le calcul de cet arbre est polynomial (problème MST). □

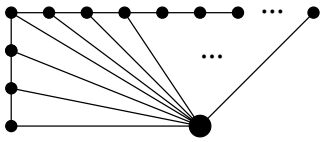
Question 10. *Montrer que le coût de l'arbre couvrant de poids minimum du sous-graphe induit par* R *est inférieur à 2.OPT (arbre de Steiner du coût minimum).*

Réponse : Soit un MST T du sous-graphe induit par R. On double ses arêtes, ce qui donne un cycle, et on court-circuite ce cycle de façon à obtenir un cycle eulérien E de R (quand on arrive sur sommet déjà visité, on se dirige directement vers le suivant, ce qui n'alourdit pas le poids total à cause de l'inégalité triangulaire). Comme vu dans le cours, $w(E) \leq 2.OPT(G)$, parce que la fonction de poids est métrique.

Le circuit E est évidemment plus lourd que T, donc $w(T) \leq 2.OPT(G)$ □

Question 11. *Trouver une instance critique pour l'algorithme d'approximation basé sur un arbre couvrant du sous-graphe induit par R. On pourra étudier la clique à $n+1$ sommets dont un seul sommet n'est pas étiqueté* ***Requis****, et où les arêtes du sous-graphe induit par R (n sommets) coûtent 2 tandis que les autres coûtent 1.*

Réponse :



Dans cette figure, tous les sommets sont dans R sauf Δ. L'arbre couvrant de R est de poids $2n-2$, alors que l'optimal est en n : la figure tend asymptotiquement à être critique. □

♣