Optimisation Combinatoire - Travaux Dirigés n° 3

— *Maîtrise d'Informatique* —

Algorithmes gloutons

Les algorithmes gloutons s'appliquent en général aux problèmes d'optimisation pour lesquels il faut effectuer un ensemble de choix avant d'arriver à une solution optimale. Le principe de l'algorithme glouton est de prendre à chaque fois l'option optimale localement sans jamais revenir sur ce choix (ce qui n'aboutit pas toujours à une solution globalement optimale). Par exemple, un arbre recouvrant de poids minimal pour un graphe complet peut être obtenu par un tel algorithme (algorithmes de Kruskal et de Prim). Il existe de nombreux problèmes de cette sorte pour lesquels une approche gloutonne fournit rapidement une solution optimale. Toutefois il n'est pas toujours facile de savoir si une approche gloutonne est efficace

1 Petite monnaie

▶ **Exercice 1**
On considère le problème consistant à rendre n centimes en utilisant le moins de pièces possibles.

1. Décrire un algorithme glouton permettant de rendre la monnaie en utilisant des pièces de 50, 20, 10, 5 et 1 centimes. Justifier le fait que l'algorithme aboutit à une solution optimale.

2. On suppose maintenant que les pièces disponibles ont pour valeurs $p^0 = 1$, p^1, …, p^k où $p > 1$ et $k \geq 1$ sont des entiers. Montrer que l'algorithme précédent conduit encore à une solution optimale.

3. Donner un ensemble de valeurs de pièces pour lesquelles l'algorithme glouton ne donne pas une solution optimale.

2 Codage de Huffman

Les codages de Huffman constituent une technique largement utilisée et très efficace pour la compression de données ; des économies de 20% à 90% sont courantes, selon les caractéristiques du fichier à compresser.

L'algorithme glouton de Huffman utilise un tableau contenant les fréquences d'apparition de chaque caractère pour une chaîne binaire et produit un code préfixe optimal appelé *codage de Huffman*. L'algorithme construit du bas vers le haut un arbre T correspondant au codage optimal. Il commence avec un ensemble de $|C|$ feuilles et effectue une série de $|C|-1$ "fusions" pour créer l'arbre final.

Dans ce qui suit, on suppose que C est un ensemble de n caractères et que chaque caractère $c \in C$ est un objet qui possédant une fréquence. Une file de priorité F permet d'identifier les deux objets les moins fréquents à fusionner. Le résultat de la fusion de deux objets est un nouvel objet dont la fréquence est la somme des fréquences des deux objets fusionnés.

HUFFMAN(C)
1 $n \leftarrow |C|$

```
F ← C
pour i ← 1 à n − 1
      faire z ← allouer-nœud()
            x ← gauche(z) ← extraire-min(F)
            y ← droite(z) ← extraire-min(F)
            f(z) ← f(x) + f(y)
            insérer (F,z)
retourner extraire-min(F)
```

Le temps d'exécution de l'algorithme de Huffman est $O(n \log n)$ sur un ensemble de n caractères.

▶ **Exercice 2** Nombres de Fibonacci

1. Donner un codage optimal pour l'ensemble des fréquences suivantes, correspondant aux 8 premiers nombres de Fibonacci:
   `a:1, b:1, c:2, d:3, e:5, f:8, g:13, h:21`.

2. Pouvez-vous généraliser votre réponse pour trouver un codage optimal lorsque les fréquences sont les n premiers nombres de Fibonacci.

▶ **Exercice 3** On considère un fichier de données contenant une séquence de caractères sur 8 bits telles que les 256 caractères apparaissent à peu près aussi souvent : la fréquence de caractère maximum vaut moins de deux fois la fréquence du caractère minimum. Prouver que l'algorithme de Huffman n'est pas plus efficace que le codage habituel de longueur fixe sur 8 bits.

▶ **Exercice 4** Démontrer que pour un codage optimal, si les caractères sont triés par ordre de fréquence décroissante, la longueur des mots est croissante.

▶ **Exercice 5** Le coût de l'arbre pour un arbre de codage T est égal à

$$B(T) = \sum_{c \in C} f(c) d_T(c)$$

où $f(c)$ et $d_T(c)$ représente respectivement la fréquence et la profondeur dans l'arbre du caractère c. Montrer que ce coût peut-être calculé comme la somme sur tous les nœuds internes des fréquences combinées des nœuds de ses fils.

▶ **Exercice 6** Mots de codes ternaires
Généraliser l'algorithme de Huffman aux mots de codes ternaires (c'est-à-dire des mots de code utilisant les symboles 0, 1 et 2) et démontrer qu'il engendre des codes ternaires optimaux.

3 Le problème du choix d'activités

Ce problème consiste à répartir une ressource parmi plusieurs activités en compétition.

Soit $S = \{1, 2, \dots, n\}$ un ensemble de n activités concurrentes qui souhaitent utiliser une ressource, comme une salle de cours, qui ne peut être utilisée que pour une activité à la fois. Chaque activité i possède un horaire de début d_i et un horaire de fin f_i, avec $d_i \leq f_i$. Si elle est choisie, l'activité i a lieu dans l'intervalle de temps ouvert $[d_i, f_i[$. Les activités i et j sont compatibles si les intervalles $[d_i, f_i[$ et $[d_j, f_j[$ ne se superposent pas (*i.e.* $d_i \geq f_j$ ou $d_j \geq f_i$). Le problème du choix d'activités est de choisir un ensemble le plus grand possible d'activités compatibles entre elles.

▶ **Exercice 7** Donner un *algorithme glouton* résolvant ce problème.

Toute *approche gloutonne* du choix d'activités ne produit pas nécessairement un ensemble maximal d'activités mutuellement compatibles.

▶ **Exercice 8** Donner un exemple illustrant que l'approche consistant à choisir l'activité la plus courte parmi celles qui sont compatibles avec les activités déjà sélectionnées n'est pas adaptée. Faire de même pour l'approche consistant à toujours choisir l'activité qui se superpose avec le moins d'autres activités restantes.

4 Coloriage d'un graphe d'intervalles

On suppose que l'on ait un ensemble d'activités à ordonnancer sur un grand nombre de salles de cours. On souhaite utiliser le moins de salles de cours possible pour qu'elles aient toutes lieu.

Donner un *algorithme glouton* permettant de déterminer quelle activité doit avoir lieu dans quelle salle de cours.

Ce problème est aussi connu sous le nom de coloriage d'un graphe d'intervalles. On peut créer un graphe d'intervalles dont les sommets sont les activités et dont les arêtes relient les activités incompatibles. Trouver le plus petit nombre de couleurs requises pour colorier chaque sommet de manière que deux sommet adjacents quelconques n'aient pas la même couleur, revient à trouver le plus petit nombre de salles nécessaires pour ordonnancer toutes les activités.

5 Le problème du havresac

Le problème du havresac est le suivant. On dipose de n objets pour remplir un *havresac*. Chaque objet i a un poids w_i et une valeur v_i et havresac a une capacité maximale de W kilos. Il s'agit de remplir le havresac avec des objets avec une valeur totale maximale mais sans pour autant dépasser la capacité du havresac.

Dans la variante *tout ou rien*, les objets sont indivisibles. Un objet est mis entièrement dans le havresac ou il est laissé complètement en dehors du havresac. Dans la variante *fractionnaire*, il est possible de mettre tout un objet ou seulemennt une fraction $0 < p_i \leq 1$ d'un objet i. Les contributions de l'objet sont alors de $p_i v_i$ en valeur et de $p_i w_i$ en poids.

▶ **Exercice 9** Donne un algorithme glouton pour la variante fractionnaire du havresac. Cette méthode fonctionne-t-elle aussi pour la variante tout ou rien ?

6 Le problème de la couverture d'ensemble

Le problème de la couverture d'ensemble est un problème d'optimisation qui modélise de nombreux problèmes de choix de ressources. C'est un problème NP-difficile. Ce qui suit présente une heuristique gloutonne simple de borne logarithmique. Autrement dit, la taille de la solution approchée par rapport à celle d'un solution optimale peut croître avec celle de l'instance.

Comme la fonction logarithme est à croissance lente, cet algorithme d'approximation peut néanmoins donner des résultats utiles.

Une instance $(X, \mathcal{F})$ du *problème de couverture d'ensemble* consiste en un ensemble fini X et une famille $\mathcal{F}$ de sous-ensembles de X, telle que tout élément de X appartient au moins à un sous-ensemble contenu dans la famille $\mathcal{F}$:

$$X = \bigcup_{S \in \mathcal{F}} S.$$

On dit qu'un sous-ensemble $S \in \mathcal{F}$ couvre ses éléments. Le problème consiste à trouver un sous-ensemble de taille minimale $\mathcal{C} \subset \mathcal{F}$ dont les membres couvrent X tout entier :

$$X = \bigcup_{S \in \mathcal{C}} S.$$

Le problème de la couverture d'ensemble est un modèle de problèmes combinatoires fréquents. Par exemple, X représente un ensemble de compétences nécessaires pour résoudre un problème. on souhaite créer un comité composé du moins de personnes possibles, tel que pour chaque compétence requise de X un membre du comité possède cette compétence. Dans la version décisionnelle de ce problème on cherche à savoir s'il existe ou non une couverture de taille au plus k, où k est un paramètre supplémentaire spécifié dans l'instance du problème. Ce problème de décision est NP-complet.

La méthode gloutonne fonctionne en choisissant, à chaque étape, l'ensemble S qui couvre le plus d'éléments non encore couverts.

```
COUVERTURE-ENSEMBLE-COMPLET(X, F)
U ← X
C ← ∅
tant que U ≠ ∅
      faire choisir S ∈ F qui maximise |S ∩ U|
            U ← U \ S
            C ← C ∪ {S}
retourner C
```

▶ **Exercice 10** On considère les mots suivants comme un ensemble de lettres
{ aride, date, drain, héron, louche, nom, short, slalom, snob, thèse}. Donner la couverture d'ensemble produite par l'algorithme précédent (si plusieurs lettres sont identiques le choix glouton se fait en faveur du mot qui apparaît le premier dans le dictionnaire).

▶ **Exercice 11** (**Problème de couverture d'ensemble pondéré**)
On suppose qu'on généralise le problème de la couverture d'ensemble de sorte que chaque ensemble S_i de la famille $\mathcal{F}$ ait un coût associé w_i et que le coût d'une couverture $\mathcal{C}$ soit égale à $\sum_{S_i \in \mathcal{C}} w_i$. On souhaite déterminer une couverture de coût minimal.

Montrer que l'heuristique gloutonne de la couverture d'ensemble peut-être généralisée de façon naturelle pour fournir une solution approchée pour toute instance du problème de la couverture d'ensemble pondéré. Montrer que l'heuristique obtenue a une borne égale à $H(d)$ où d est la taille maximale d'un ensemble S_i.