École polytechnique
Promotion 2004

Majeure informatique

Cours de conception et analyse d'algorithmes
PC 7 – algorithmes probabilistes
7 mars 2007

0. Programmation linéaire, algorithmes probabilistes

On considère le problème SET-COVER pour lequel la donnée est un ensemble $E = \{1, 2, \ldots, n\}$ et une famille $\mathcal{F}$ de sous-ensembles $F_1, F_2, \ldots, F_m$ de E dont l'union est égale à E. On suppose de plus que chaque élément i est contenu dans exactement k sous-ensembles F_i. Il s'agit de trouver un nombre minimum de sous-ensembles $F_i, i \in I$ tels que $\bigcup_{i \in I} F_i = E$.

On se propose ici d'utiliser la programmation linéaire et un algorithme probabiliste pour résoudre ce problème. Pour cela on associe à chaque F_i une variable x_i qui prendra des valeurs comprises entre 0 et 1.

i. Exprimer le problème sous forme d'un programme linéaire en nombres entiers comportants n contraintes (en plus de $0 \leq x_i \leq 1$) et une fonction à minimiser. Ecrire ces contraintes pour l'exemple suivant $E = \{1, 2, 3, 4, 5\}$ et $F_1 = \{1, 3, 5\},\ F_2 = \{2, 3, 4\},\ F_3 = \{1, 2\},\ F_4 = \{4, 5\},\ F_5 = \{1, 2, 4\},\ F_6 = \{3, 5\}$.

ii. Soit $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m$ la solution au problème lorsqu'on relache la condition nombres entiers pour autoriser des rationnels. Donner une borne inférieure pour l'optimum du cas entier en fonction de la solution $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m$.

iii. On applique la méthode suivante pour résoudre notre problème à partir de la solution en nombres rationnels $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m$: choisir si $i \in I$ avec probabilité $\hat{x}_i$. Cette méthode est-elle de type Monte-Carlo ou Las Vegas ?

Quelle est en fonction des $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m$ la probabilité pour qu'un élément j donné appartenant à $F_{j_1}, F_{j_2}, \ldots, F_{j_k}$ n'appartienne à aucun des F_i pour $i \in I$?

iv. Montrer que la probabilité que j soit dans un des $F_i, i \in I$ est minorée par $1 - \frac{1}{e}$.

v. Donner un algorithme qui après avoir résolu le problème linéaire en nombres rationnels effectue $m \log(n)$ tirages aléatoires pour donner une solution $O(\log n)$- approchée et ceci avec probabilité supérieure ou égale à $\frac{1}{2}$.

1. Un algorithme probabiliste pour 2-SAT

Une formule 2-SAT consiste en p clauses $u_i = y_i \vee z_i$ sur n variables. On considère l'algorithme probabiliste suivant pour chercher une assignation satisaisant la formule (m étant un paramètre) :

- partir d'une affectation quelconque de valeurs aux variables $x = (x_1, \ldots, x_n)$.
- pour i de 1 à m faire
  s'il existe une clause c non satisfaite,
    changer la valeur d'une des deux variables choisie au hasard
  sinon renvoyer "satisfiable"

– renvoyer “non satisfiable”

i. L'algorithme est-il de type Monte Carlo ou Las Vegas ? Préciser. Quelle est la probabilité d'erreur si la formule est non satisfiable ?

On suppose qu'il existe une affectation u satisfaisant toutes les clauses et on oublie momentanément la condition d'arrêt.

ii. Soit $E(x)$ le nombre moyen d'itérations avant de renvoyer “satisfiable” en partant d'une assignation x, et soit $E(i)$ le maximum des $E(x)$ sur toutes les assignations x telles que la distance de Hamming $d(x, u)$ soit inférieure ou égale à i. Montrer que :

$$\begin{cases} E(0) & = & 0 \\ E(n) & \leq & 1 + E(n-1) \\ E(i) & \leq & 1 + (E(i-1) + E(i+1))/2 \text{ pour } 1 \leq i \leq n-1. \end{cases}$$

iii. En déduire successivement que $E(i) \leq 2(n-i+1) - 1 + E(i-1)$ pour tout $i \in \{1 \ldots, n-1\}$, puis une borne supérieure sur $E(n)$.

iv. Par l'inégalité de Markov, en déduire que si $m > 2n^2$, alors la probabilité d'obtenir l'affectation correcte est au moins égale à $1/2$.

v. Que penser d'une généralisation à 3-SAT ?

2. Coupure minimale

Soit G un graphe connexe non orienté, dont les arêtes peuvent être multiples, avec n sommets. Une *coupure* dans G est un ensemble d'arêtes dont la suppression rend G non connexe. Une coupure minimale est une coupure de cardinalité minimale. Voici un algorithme probabiliste pour la détermination d'une coupure minimale.

Contracter $\{u, v\}$ signifie : fusionner ces deux sommets en un seul sommet w ; les arêtes incidentes à w sont les arêtes qui étaient précédemment incidentes à u ou à v ; puis, supprimer les boucles (induites par les arêtes $\{u, v\}$).

L'algorithme contracte une suite d'arêtes choisies au hasard, jusqu'à ce que le graphe n'ait plus que deux sommets ; il renvoie alors comme valeur de coupure le nombre d'arêtes entre ces deux sommets.

i. Montrer que l'opération de contraction ne diminue pas la valeur de la coupe minimale.

ii. Soit k la taille de la coupe minimale. Montrer que G a au moins $kn/2$ arêtes.

iii. Quelle est la probabilité qu'après ℓ étapes, aucune des arêtes d'une coupe minimale C de taille k n'ait été contractée ? En déduire une borne inférieure sur la probabilité que l'algorithme renvoie la valeur d'une coupe minimale.

iv. Quelle est la complexité de l'algorithme ? Construire un algorithme polynomial renvoyant la bonne valeur avec probabilité $1 - O(1/n^2)$.

3. Vérification de calculs matriciels

L'algorithme le plus efficace pour la multiplication de deux matrices carrées de côté n actuellement connu est de complexité $O(n^{2.376})$. Hélas, il est extrêmement complexe, et toute implémentation est sujette à erreurs.

On souhaite tester le programme en vérifiant sur un certain nombre de couples (A, B) si le résultat C fourni est bien le bon.

i. Quelle est la complexité du calcul de Ax pour un vecteur x donné, par une méthode naïve ?

ii. Soit r un vecteur à coordonnées 0 ou 1 choisi au hasard. On suppose que $AB \neq C$. Montrer que la probabilité que $ABr = Cr$ est au plus $1/2$.

iii. Comment et en quelle complexité peut-on être convaincu que $AB = C$ avec probabilité $1 - O(1/n)$?

4. Stratégies de cache

On s'intéresse aux *stratégies de cache* déterministes et probabilistes. Dans les ordinateurs, il existe au moins deux niveaux de mémoire physique : le cache, petit mais d'accès rapide ; et la RAM, beaucoup plus grande mais d'accès plus lent. Le problème, étant donné une suite de requêtes, est de choisir à tout instant quelles données garder dans le cache. Soit k le nombre d'unités de mémoire du cache. À chaque fois qu'une requête sollicite une information non présente, il faut la charger dans le cache, et l'algorithme doit pour cela choisir une information à éliminer du cache.

On considère un algorithme O optimal en ce sens qu'il a connaissance de toute la suite des requêtes futures et peut ainsi choisir sa stratégie d'élimination en conséquence. On va comparer les performances de O à celles d'un algorithme déterministe quelconque A, puis à celle d'un algorithme probabiliste donné **Marker**.

Pour cela on va considérer chaque requête comme une lettre de l'alphabet (une lettre différente par page mémoire demandée) ainsi la donnée de l'algorithme est un mot f représentant la suite des requêtes.

4.1

Quel est le nombre minimum de transferts du cache vers la mémoire pour un cache composé de 3 emplacements dans le cas où le mot d'entrée est

$$abdabecbcabadbac$$

lorsque le cache contient a, b, c au départ.

4.2 Donner un exemple de mot d'entrée tel que pour une mémoire cache de k places l'algorithme optimal fait 1 transfert alors qu'un algorithme qui ne connaît pas la totalité du mot peut en faire k.

On considère la stratégie probabiliste **Marker** suivante. À chaque emplacement du cache, on associe un bit initialement mis à 0. Pour chaque requête, on opère de la façon suivante :

- si la page cherchée est en cache, on met le bit correspondant à 1.
- si la page n'est pas en cache, on la charge, et on rejette une page à un emplacement choisi au hasard dans le cache *parmi ceux dont le bit est à 0*, et on met le bit de cet emplacement à 1.

Une fois que les bits de tous les emplacements sont à 1, on les remet tous à 0 et on recommence.

On note $f_1 f_2 \ldots f_m$ le mot d'entrée. Une *étape* est un facteur du mot d'entrée entre deux remises à 0 des bits.

4.3. Montrer qu'une étape, contient exactement k lettres distinctes..

4.4 On considère une étape : $f_p f_{p+1} \ldots f_q$ de l'algorithme **Marker**, on dit qu'un indice $i, p \leq i \leq q$ est actif si la lettre f_i n'était pas dans le cache de **Marker** au début de l'étape.

Soit ℓ le nombre d'indices actifs lors de l'étape considérée, d_I le nombre d'objets du cache de O qui ne sont pas dans le cache de **Marker** en début d'étape, et d_F l'analogue en fin d'étape.

Montrer que le nombre de transferts de O est au moins $\max(\ell - d_I, d_F)$, donc $(\ell - d_I + d_F)/2$. En déduire que le nombre de transferts au total est au moins $(L-k)/2$ où L désigne le nombre total d'indices actifs.

4.5. Rappelons qu'un indice est actif si la lettre f_i n'était pas dans le cache de **Marker** au début de l'étape. Inversement on dit qu'il est inactif s'il y était.

On s'intéresse toujours à une étape aucours de laquelle ℓ indices actifs vont être rencontrés. On se place maintenant au cours du déroulement de l'étape, alors qu'on a rencontré j indices actifs et i indices inactifs. Montrer que la probabilité qu'une lettre correspondant à un indice inactif (non encore requis) soit présente dans le cache lors de son examen est $(k-i-j)/(k-i)$. En déduire que si on veut insérer la lettre à ce moment là, la probabilité de ne pas la trouver est $j/(k-i)$.

En déduire que le pire des cas pour une étape avec ℓ indices actifs est : d'abord ℓ indices actifs, puis $k-\ell$ indices inactifs.

d. En déduire qu'en moyenne **Marker** effectue au plus $2\sum_{j=1}^{k} 1/j = O(\log k)$ transfert quand O en effectue 1.