Nom : Hany Daphné Sujet : 11 Jury : Page : 1 / 3

Leçon 11 : Exemples d'algorithmes d'approximation et d'algo. probabilistes

Niveau : MP2I/MPI

Motivation 1 : Beaucoup de problèmes sont durs à résoudre. On veut alors sacrifier la fiabilité du résultat pour gagner en complexité. On propose ici 2 manières de faire un tel compromis :

- Utiliser des algorithmes probabilistes (pouvant se tromper ou prendre un temps non déterministe) (I)
- Utiliser des algorithmes qui fournissent une solution approchée (II)

I - Algorithmes probabilistes

Def 2 : Un algorithme est déterministe si, pour une entrée x donnée, il s'exécute toujours de la même manière. En particulier, sa sortie est fonction de l'entrée.

Un algorithme est probabiliste si son exécution dépend de son entrée x et de valeurs obtenues via un générateur de nombre pseudo-aléatoire.

Exemple 3 : Recherche dans un tableau de booléens T un indice i tq $T[i] == true$

```
def deterministe (T) :
    pour i de 1 à |T| :
        Si T[i] :
            return true
    return false
```

```
def probabiliste (T, k) :
    pour j de 1 à k :
        tirer uniformément i dans [1,n]
        Si T[i] :
            return true
    return false
```

Remarque 4 : Soit $p<1$ la proportion de booléens à False dans T. L'algo probabiliste se trompe avec proba p^k.

Si $p = 1/4$ et $k = 5$, cela fait 10^{-3}

I.1 - Algorithmes de type Monte-Carlo

Def 5 : Un algorithme probabiliste A pour un pb Π est de type Monte Carlo si $\forall$ instance i de Π :

- $A(i)$ est une solution, erronée avec une certaine probabilité
- Le temps d'exécution de A sur i est indépendant des choix aléatoires.

Exemple 6 : l'algorithme probabiliste de l'exemple 3 est de type Monte-Carlo.

Développement 1 : Vérification probabiliste du produit matriciel

Proposition 7 : (Amplification) Soit $0 < \varepsilon_2 < \varepsilon_1 < 1$. S'il existe un algorithme Monte-Carlo pour un pb Π avec une probabilité d'erreur ε_1, alors on peut construire un algorithme Monte Carlo pour Π avec probabilité d'erreur ε_2, en appliquant plusieurs fois le premier.

Algo 8 : Calcul probabiliste de la médiane

```
mediane (L) :
    n = |L|
    selectionner k = n^(3/4) éléments de L aléatoirement
    les trier
    x1 <- le k/2 - √n ième élement
    x2 <- k/2 + √n ième élement
    séparer L en : L1 : liste d'elements < x1
                   L2 : liste d'elements entre x1 et x2
                   L3 : liste d'elements > x2
    Si |L1| > n/2 ou |L3| > n/2 :
        renvoyer n'importe quoi
    Si |L2| > n^(3/4) : renvoyer n'importe quoi
    Sinon : trier L2 ; renvoyer L2 [n/2 - |L1|]
```

Ne rien écrire hors du cadre.

I.2 - Algorithmes de type Las Vegas

Déf 9: Un algorithme probabiliste A est de type Las Vegas si:
* Si A termine, la solution renvoyée est correcte.
* le temps d'exécution de A est une variable aléatoire.

Algorithme 10:

```
Tri rapide randomisé (T)
    Si T de taille 1: retourner
    tirer q uniformément dans [1, |T|] // T[q] sera le pivot
    i = partition(T, q)
    tri rapide randomisé (T[:i-1])
    tri rapide randomisé (T[i+1:])
```

Remarque 11: Le temps d'execution de l'algorithme 10 dépend du choix du pivot: $O(n^2)$ dans le pire des cas, $O(n\log n)$ dans le meilleur des cas. En moyenne, on peut montrer $O(n \log n)$.

Remarque 12: Il est possible de transformer un algorithme de type Las Vegas en Monte Carlo en l'executant pendant un temps déterminé et en générant une réponse aléatoire s'il n'a pas terminé.

Remarque 13: Pour l'algorithme de la médiane, on peut faire la transformation inverse: à la place de l'échec, on relance l'algorithme.

Remarque 14: On obtient alors un algorithme qui trouve toujours la médiane, en moyenne en $1{,}5n$ comparaisons.

II - Algorithmes d'approximation

II.1 - Définitions

Déf 15: Un problème d'optimisation Π est un triplet (I, S, c) tq:
- I est l'ensemble des instances
- $\forall i \in I$, $S(i)$ est l'ensemble des solutions réalisables.
- $c : I \times S \to \mathbb{R}$ est une fonction d'évaluation

Le but est, pour chaque instance $i \in I$, de trouver $s^* \in S(i)$ tq $c(i, s^*) = \min \{c(i,s) \mid s \in S(i)\}$. On note $OPT(i) = c(i, s^*)$.

Remarque 16: On se ramène à un pb de maximisation en prenant $-c$.

Déf 17: Une α-approximation est un algorithme A polynomial donnant pour chaque instance i de Π une solution $A(i)$ tq:

$$\max\left(\left|\frac{A(i)}{OPT(i)}\right|, \left|\frac{OPT(i)}{A(i)}\right|\right) \leq \alpha.$$

II.2 - Exemples

Problème 18: Couverture par des sommets (Vertex-Cover)

Entrée: Un graphe $G = (V, E)$

Solution: un ensemble $S \subset V$ tq $\forall (u,v) \in E$, $v \in S$ ou $u \in S$

Algorithme 19: Glouton Vertex Cover:

```
S = {}
Tant qu'il existe une arête (u,v) ∈ E tq u ∉ S, v ∉ S:
    Choisir (u,v) une telle arête
    S = S ∪ {u,v}
renvoyer S
```

Propriété 20: L'algorithme 19 est une 2-approx du problème Vertex-Cover.

Developpement 2: Une 3/2-approximation gloutonne pour le pb d'ordonnancement des tâches indépendantes sur 2 processeurs

Problème 21 : Voyageur de Commerce (TSP)

Instance : $G=(V,E)$ un graphe complet orienté ; $c : E \to \mathbb{N}$

Solution : Un cycle hamiltonien sur G de poids minimal.

Théorème 22 : Il n'existe pas d'algorithme d'approximation pour le pb de Voyageur de Commerce, sauf si $P=NP$.

Problème 22 : Voyageur de commerce avec inégalité triangulaire

Instance : $G=(V,E)$ graphe complet ; $c : E \to \mathbb{N}$ tq $\forall (u,v),(v,w) \in E^2$, $c(u,w) \leq c(u,v) + c(v,w)$.

Solution : Un cycle hamiltonien sur G de poids minimal.

Algorithme 23 : heuristique (G,c) :

1 A = arbre couvrant de poids minimal sur G

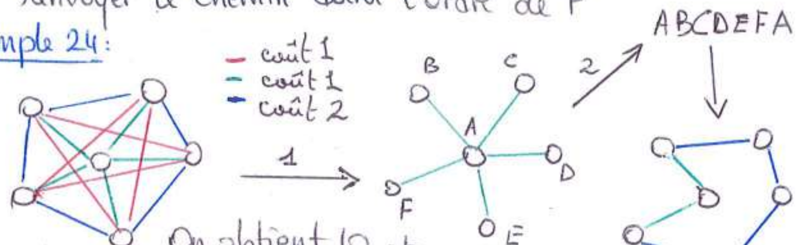
2 P = parcours en préfixe de A enraciné

//P peut passer plusieurs fois par le meme noeud

//$c(P) = 2\,c(A)$

3 renvoyer le chemin dans l'ordre de P

Exemple 24 :



On obtient 10 et non 6.

Propriété 25 : l'algorithme 23 est une 2-approximation pour TSP avec inégalité triangulaire.

III - Algo d'approximation probabiliste

Il existe des algorithmes d'approximation probabilistes. Ce sont alors souvent des algorithmes de type Monte-Carlo.

III. 1 - Max-SAT

Problème 26 : MAX-SAT

Instance : n variable $(x_1, \dots, x_n)$, p clauses $C_1, \dots, C_p$ sur $(x_1, \dots, x_n)$ contenant au moins k littéraux

$\varphi = \bigwedge_{i=1}^{p} C_i$

Solution : trouver une valuation σ qui minimise le nombre de clauses à FAUX.

Exemple 27 : Si φ est satisfaisable, une solution optimale est une valuation satisfaisant φ.

Algo 28 : renvoyer une valuation aléatoire.

Théorème 29 : L'espérance du nombre de clauses satisfaite $\mathbb{E}(\varphi)$ vérifie $\mathbb{E}(\varphi) \geq p\left(1 - \frac{1}{2^k}\right) \geq \frac{c}{2}$

Remarque 30 : On peut créer un algo déterministe qui est une $\left(1 - \frac{1}{2^k}\right)$ approx.

III. 2 - Calcul de π

Algorithme 31 :

calcul-pi (N) :

i=0

répéter N fois :

tirer deux flottants x, y dans $[-1,1]$ uniformément

Si $x^2 + y^2 \leq 1$: // On a tapé dans le cercle d'aire π

i += 1

renvoyer $4i/N$ // nbr de points dans le cercle / aire du carré $= \frac{\pi}{4}$

Remarque 32 : On peut généraliser la methode pour le calcul d'intégrales

Remarque 33 : Cette méthode se généralise en une méthode de Monte-Carlo où on échantillonne le problème, on calcule la solution sur cet échantillon, puis on généralise.