11 - Exemples d'algorithmes d'approximation et d'algorithmes probabilistes

Niveau : MPI Prérequis : Analyse d'algorithme, Complexité (classe P et NP)

Motivation : Certains problèmes difficiles ne peuvent être résolus rapidement par des algorithmes exacts. Il est possible d'y pallier au prix par exemple de l'exactitude, notamment grâce à des algorithmes d'approximation et/ou probabilistes.

I] Algorithmes d'approximation

1) Définitions

Définition 1 : Un problème d'optimisation consiste en la recherche, parmi un ensemble de solutions S du problème, d'une solution $s \in S$ minimisant/maximisant une fonction de coût c.

Exemples 2 : Problème du sac à dos, recherche d'un plus court chemin, etc...

Définition 3 : On dit qu'un algorithme d'approximation a une garantie de performance $\rho(n)$ si pour toute entrée de taille n, le coût C de la solution renvoyée par l'algorithme est éloignée au plus d'un facteur $\rho(n)$ du coût de la solution optimale C^*. i.e. $\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$
Dans ce cas, il s'agit d'un algorithme d'approximation $\rho(n)$

2) Exemples d'algorithmes d'approximation

a) Problème de la couverture de sommets

Définition 4 : Une couverture de sommets d'un graphe non-orienté $G=(S,A)$ est un sous-ensemble $S' \subseteq S$ tel que pour toute arête $(u,v) \in A$, on a $u \in S'$ ou $v \in S'$

Définition 5 : Le Problème de la couverture de sommets consiste à trouver une couverture de sommets S' de taille minimale dans un graphe G non orienté.

Propriété 6 : Le problème de décision associé au problème de la couverture de sommets est NP-complet.

Propriété 7 : COUVERTURE-SOMMET(G) renvoie une couverture de sommets de G en temps polynomial.

```
COUVERTURE-SOMMET(G):
    C ← ∅
    A' ← A[G]
    Tant que A' ≠ ∅ :
        Soit (u,v) ∈ A'
        C ← C ∪ {u,v}
        Supprimer de A' les arêtes
        incidentes à u ou v
    Retourner C.
```

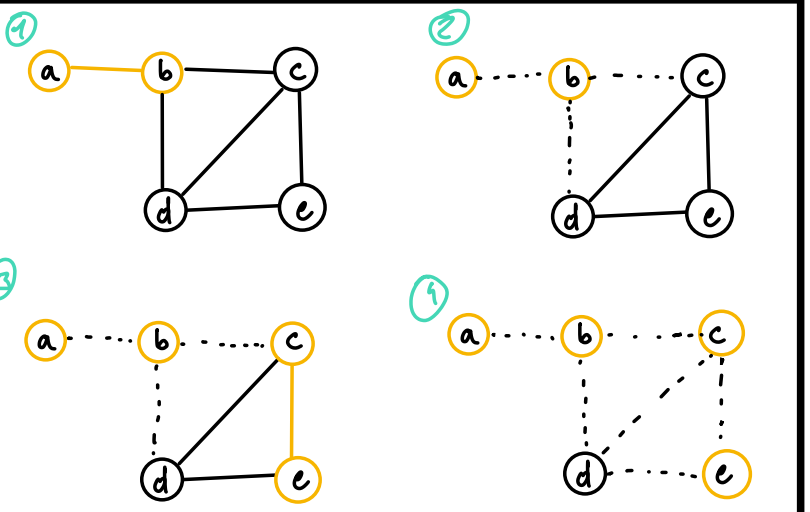


Figure 8 : Illustration d'une exécution de COUVERTURE SOMMET

COUVERTURE-SOMMET choisit une arête pas encore couverte et ajoute les deux sommets de celle-ci à la couverture courante, jusqu'à ce que toutes les arêtes soient couvertes. On se permet donc de parler des arêtes de la couverture renvoyée par COUVERTURE-SOMMET.

Propriété 9 : ① les arêtes choisies lors de l'exécution de COUVERTURE-SOMMET sont disjointes. ② Soit S une couverture de G, pour toute arête (u,v) choisie par COUVERTURE-SOMMET, on a $u \in S$ ou $v \in S$.

Théorème 10 COUVERTURE-SOMMET est un algorithme d'approximation 2 en temps polynomial pour le problème de la couverture de sommets.

b) Voyageur de commerce avec inégalité triangulaire

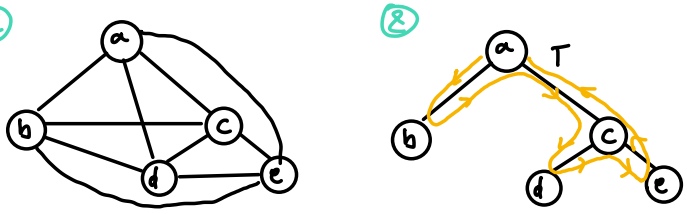
Définition 11 : Le problème du voyageur de commerce consiste à trouver un cycle hamiltonien de coût minimal dans un graphe G complet, $c(u,v)$ étant le coût de l'arête (u,v). Si on a $\forall u,v,w \in S$, $c(u,w) \leq c(u,v) + c(v,w)$ alors on dit que G vérifie l'inégalité triangulaire. On parle de problème du voyageur de commerce avec inégalité triangulaire.

Propriété 12 : Soit T un arbre couvrant de G de poids minimal. Soit H^* un cycle hamiltonien minimal de G. On a $c(T) \leq c(H^*)$

Propriété 13 : Soit P un parcours complet de T. on a $c(P) = 2 \cdot c(T)$ et tout cycle hamiltonien H extrait de P vérifie $c(H) \leq c(P) = 2 \cdot c(T)$

Théorème 14 : VC est un algorithme d'approximation 2 à temps polynomial pour le problème du voyageur de commerce avec inégalité triangulaire.

```
VC(G)
    Sélectionner r ∈ S
    Calculer un ACM T de G
        ayant pour racine r
    Soit L un parcours préfixe de T
    renvoyer L
```



parcours complet: a, b, a, c, d, c, e, c, a
parcours 2-optimal: a, b, –, c, d, –, e, –, –

Figure 15: Illustration d'une exécution de VC.

Remarque 16: Dans le cas général du problème du voyageur de commerce, on sait qu'il n'existe pas d'algorithme d'approximation ayant une garantie de performance constante !

c) Problème de couverture d'ensembles

Définition 17: Une couverture d'un ensemble X est un ensemble C de parties de X tel que $\bigcup_{S \in C} S = X$.

Définition 18: Le problème de la couverture d'ensembles consiste à déterminer une couverture minimale de X n'utilisant que des parties dans un ensemble F.

Théorème 19: Un algorithme glouton est une $(\ln|X| + 1)$-approximation en temps polynomial pour la couverture d'ensembles. DEV 1

3) Schéma d'approximation

On peut parfois faire un compromis entre garantie de performance et temps d'exécution.

Définition 20: Un schéma d'approximation est un algorithme d'approximation prenant en entrée non seulement une instance du problème mais aussi un paramètre $\varepsilon > 0$, telle que, pour ε fixé, l'algorithme d'approximation ait une garantie de performance d'au moins $(1+\varepsilon)$.
Si de plus, pour ε fixé, l'algorithme s'exécute en temps polynomial, alors on l'appelle schéma d'approximation polynomial.
Si de plus le temps d'exécution est polynomial en $\frac{1}{\varepsilon}$ et en n, la taille de l'entrée, alors on dit du schéma qu'il est entièrement polynomial.

Activité 21: Examen/DM sur un schéma d'approximation entièrement polynomial pour le problème de la somme de sous-ensembles.

II] Algorithmes probabilistes

1) Définition et premier exemple

Définition 22: Un algorithme probabiliste est un algorithme qui utilise une source de hasard, comme par exemple un générateur de nombre pseudo-aléatoires.

Un premier exemple d'algorithme probabiliste est la version randomisée du tri rapide.

```
TRI-RAPIDE(A,p,r):
  Si p<r:
    q ← PARTITION-R(A,p,r)
    TRI-RAPIDE(A,p,q-1)
    TRI-RAPIDE(A,q+1,r)
```

```
PARTITION-R(A,p,r):
  i ← Random(p,r)
  A[i] ⟷ A[r]
  PARTITION(A,p,r)
```

```
PARTITION(A,p,r):
  x ← A[r]
  i ← p-1
  Pour j allant de p à r-1:
    Si A[j] ≤ x:
      i ← i+1
      A[i] ⟷ A[j]
  A[i+1] ⟷ A[r]
  retourner i+1
```

Théorème 23: TRI-RAPIDE(A,p,r) trie le tableau A[p..r], et effectue en moyenne $O(n \log n)$ comparaisons.

2) Classification d'algorithmes probabilistes

Définition 24: Un algorithme de Monte-Carlo est un algorithme probabiliste qui donne un résultat correct avec certaine probabilité, et termine toujours.

Définition 25: Un algorithme de Las-Vegas est un algorithme probabiliste dont le temps d'exécution n'est pas déterministe (i.e dépend de la source d'aléatoire) et donne toujours le bon résultat.

Remarque 26: L'algorithme TRI-RAPIDE donne toujours le bon résultat, et son temps d'exécution varie entre $O(n \log n)$ et $O(n^2)$, C'est donc un algorithme de Las Vegas. Attention, un algorithme de Las Vegas peut ne pas toujours terminer!

3) Exemples d'algorithmes probabilistes

a) Test d'égalité de polynômes.

Définition 27 : le problème du test d'égalités de polynômes consiste à déterminer si deux polynômes P et Q donnés en entrée sont égaux. P étant donné sous forme factorisée et Q sous forme développée.

Une approche naïve consiste à développer le polynôme P et tester l'égalité terme à terme. Mais cet approche a une complexité en $O(d^2)$ où d est le degré maximal de P et Q. Une version randomisée peut être plus efficace.

```
EGALITE (P, Q):
    d ← max(deg(P), deg(Q))
    i ← Random (0, 100.d)
    Renvoyer P(i) == Q(i)
```

$P(x) = (x-3)^2$

$Q(x) = x^2 - 10x + 25$

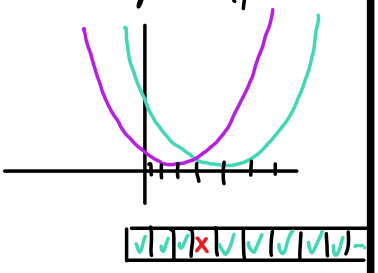


Figure 28 : Exemple d'un faux positif pour le test d'égalité de polynômes

Propriété 28 :
1. Si P = Q, alors EGALTE (P, Q) renvoie le bon résultat
2. Si P ≠ Q, alors EGALITE(P, Q) renvoie le bon résultat avec probabilité d'au moins 99/100.
3. EGALITE (P, Q) s'éxécute en temps linéaire.

Remarque 29 : EGALITE(P, Q) est un algorithme de Monte-Carlo qui se trompe avec probabilité au plus q = 1/100. On peut exécuter plusieurs fois cet algorithme sur la même entrée pour diminuer la probabilité d'erreur.

b) Test de primalité

On cherche un algorithme qui détermine rapidement si un entier n donné en entrée est premier ou non.

Propriété 30 : Un entier $n \in \mathbb{N}^*$ est premier si et seulement si $\forall a \in [\![2, n-1]\!]$, $a^{n-1} \equiv 1 [n]$ (Théorème de Fermat)

```
TEST-PREMIER (n):
    a ← Random (2, n-1)
    Si a^(n-1) ≡ 1 [n]:
        renvoyer Vrai
    Sinon:
        renvoyer Faux
```

Théorème 31 Si TEST-PREMIER (n) renvoie Faux, alors n est composé. Sinon n est "probablement" premier.

Remarque 32 : TEST-PREMIER est un algorithme de Monte-Carlo dont on ne connaît à priori pas le taux de succès.

c) Calcul de la Médiane

Présentation, Correction et complexité d'un algorithme probabiliste pour le calcul de la médiane d'une liste. DEV 2

4) Algorithme de génération aléatoire

Tous les algorithmes probabilistes ne sont pas des algorithmes de Monte-Carlo ou de Las-Vegas. On peut par exemple générer des objets / structures aléatoires.

a) Mélange de Knuth

Principe : On cherche à mélanger une liste, de sorte à ce que toutes les permutations aient la même probabilité d'être renvoyée.

```
MELANGE(L):
    n ← longueur(L)
    Pour i allant de 1 à n-1:
        j ← Random(i, n)
        A[i] ⟷ A[j]
```

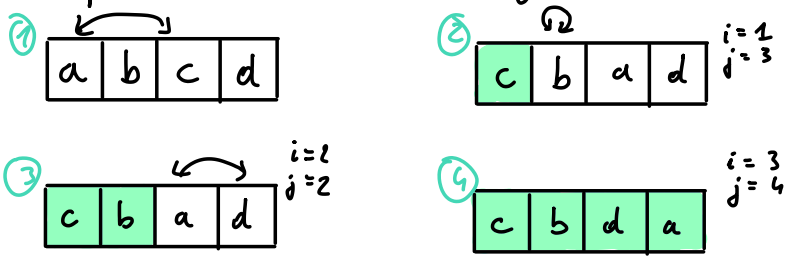


Figure 33 : Illustration d'un mélange de Knuth

Théorème 34 : MELANGE(L) renvoie une permutation quelconque de L avec probabilité 1/n!

b) Génération de grands nombres premiers.

Comme le test de primalité de Fermat fonctionne particulièrement bien sur les grands nombres, on en déduit un algorithme qui génère de très grands nombres premiers (plusieurs centaines de chiffres)

```
GENERE-PREMIER(k):
    Faire :
        n ← Random(2^k, 2^(k+1) - 1)
    Tant que :
        Non(TEST-PREMIER(n))
    Renvoyer n
```