

Algorithmique — 2. Techniques algorithmiques

7. Algorithmes probabilistes

Bruno Grenet



<https://membres-ljk.imag.fr/Bruno.Grenet/Algorithmique.html>

Université Grenoble Alpes – IM²AG
L3 Mathématiques et Informatique

Table des matières

1. Qu'est-ce qu'un algorithme probabiliste ?

2. Exemple 1 : coupe minimale

3. Exemple 2 : tri rapide

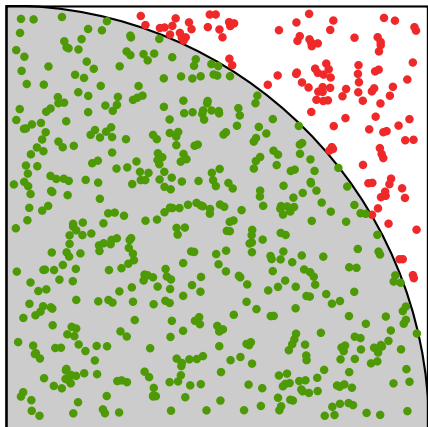
Table des matières

1. Qu'est-ce qu'un algorithme probabiliste ?

2. Exemple 1 : coupe minimale

3. Exemple 2 : tri rapide

Premier exemple : calculer π



MONTECARLO(N):

1. $c \leftarrow 0$
2. Répéter N fois :
3. $(x, y) \leftarrow$ réels aléatoires uniformes dans $[0, 1]$
4. Si $x^2 + y^2 \leq 1$: $c \leftarrow c + 1$
5. Renvoyer c/N

Propriété

$$\mathbb{E}[c/N] = \frac{\text{aire du } \frac{1}{4} \text{ de disque}}{\text{aire du carré}} = \frac{\pi/4}{1} = \frac{\pi}{4}$$

Remarques

- L'algorithme permet de calculer π
 - Mais il faut savoir tirer aléatoirement des réels
 - Et c'est une *mauvaise* façon de le faire

Deuxième exemple : trouver un nombre premier

Énoncé du problème

Entrée : entier N

Sortie : un nombre premier $p \geq N$

Échantillonnage par rejet

1. $p \leftarrow 0$
2. Tant que $\text{NON}(\text{ESTPREMIER}(p))$:
3. $p \leftarrow$ entier (impair) uniforme $\in [N, 2N]$
4. Renvoyer p

Propriétés

- ▶ Hypothèse : on dispose d'un algorithme ESTPREMIER
- ▶ L'algorithme renvoie toujours un nombre premier
- ▶ L'espérance du nombre d'entiers tirés est $O(\log N)$
 - ▶ Théorème des nombres premiers $\Rightarrow \#\{p \in [N, 2N] : p \text{ premier}\} = \Omega(N/\log N)$

Troisième exemple : vérifier un produit de matrices

Énoncé du problème

Entrées : trois matrices $A, B, C \in \mathbb{K}^{n \times n}$

Sortie : est-ce que $A \cdot B = C$?

Objectif : Ne pas faire le produit $A \cdot B$

coût : $O(n^3)$ ou $O(n^\omega)$

Algorithme de Freivalds

1. $\vec{v} \leftarrow$ vecteur aléatoire uniforme $\in \{0, 1\}^n$
2. $\vec{x} \leftarrow A \cdot (B \cdot \vec{v})$
3. $\vec{y} \leftarrow C \cdot \vec{v}$
4. Renvoyer $\vec{x} \stackrel{?}{=} \vec{y}$

Propriétés

- ▶ Complexité : 3 produits matrice-vecteur $\rightarrow O(n^2)$
- ▶ Correction :
 - ▶ Si $A \cdot B = C$, l'algo. renvoie VRAI
 - ▶ Sinon : on peut montrer que $\Pr[A \cdot B \cdot \vec{v} = C \cdot \vec{v}] \leq \frac{1}{2} \rightarrow$ renvoie FAUX avec proba. $\geq \frac{1}{2}$

Définition

Un **algorithme probabiliste** est un algorithme qui effectue des **choix aléatoires** au cours de son exécution.

Remarque

- Choix aléatoires : accès à un générateur de bits aléatoires, ou d'entiers aléatoires, etc.

Analyse d'un algorithme probabiliste

- Le comportement d'un algorithme probabiliste est une *expérience probabiliste*
- Le résultat renvoyé (*correction*) ou le nombre d'opérations effectuées (*complexité*) peuvent dépendre des choix aléatoires (parfois les deux)

Trois types d'algorithmes

1. Approximation numérique
2. Toujours correct mais temps variable
3. Temps fixé mais parfois incorrect

*calcul de π
trouver un premier
vérifier un produit*

Les algorithmes numériques

Algorithme probabiliste numérique

- ▶ Calcul d'une *valeur approchée* d'une quantité
- ▶ Souvent par échantillonnage

Analyse de l'algorithme

- ▶ Qualité de l'approximation
- ▶ En fonction du temps passé

Cadre plus général

- ▶ Algorithme d'approximation : calculer une valeur approchée
- ▶ Pas forcément de manière probabiliste !

cf. Cours 9

Remarques

- ▶ Algorithmes souvent rencontrés en maths applis

Les algorithmes de type *Las Vegas*

Las Vegas: « toujours correct, souvent rapide »

Algorithme probabiliste dont

- ▶ le résultat ne dépend pas des choix aléatoires
- ▶ la complexité dépend des choix aléatoires

Analyse de l'algorithme

- ▶ Correction : standard
- ▶ Complexité :
 - ▶ Espérance du nombre d'opérations, modélisé par une variable aléatoire
 - ▶ *Bornes de concentration*: « la probabilité que la complexité dépasse $10n^2$ est $\leq \frac{1}{n}$ »

Signification de l'espérance de la complexité

« L'espérance de la complexité est $O(n^2)$ »

- ▶ On s'attend à avoir une exécution en temps $O(n^2)$
- ▶ Si on exécute plusieurs fois l'algorithme, le temps de calcul moyen sera $O(n^2)$

Les algorithmes de type *Monte Carlo*

Monte Carlo: « toujours rapide, souvent correct »

Algorithme probabiliste dont

- ▶ la complexité ne dépend pas des choix aléatoires
- ▶ le résultat dépend des choix aléatoires

Analyse de l'algorithme

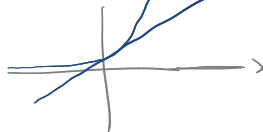
- ▶ Complexité : standard
- ▶ Correction : *probabilité de succès* (l'algorithme renvoie un résultat correct)

Améliorer la probabilité de succès

Si la probabilité de succès est p et qu'on répète N fois l'algorithme

- ▶ probabilité qu'aucune répétition ne soit correcte est $(1 - p)^N \leq e^{-pN}$

$$\forall x \in \mathbb{R} \quad 1 + x \leq e^x$$



Liens entre les différents types d'algorithmes

Numérique : cas particulier de *Monte Carlo*

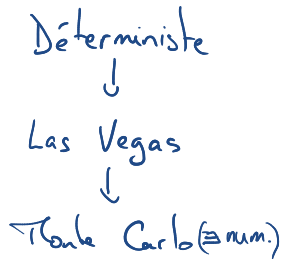
- ▶ On veut estimer une quantité α à $\pm 1\%$
- ▶ Algorithme correct si l'estimation renvoyée $\in [0.99 \cdot \alpha, 1.01 \cdot \alpha]$

Las Vegas : définition alternative

- ▶ Algorithme probabiliste dont
 - ▶ la complexité ne dépend pas des choix aléatoires
 - ▶ le résultat est soit correct soit ÉCHEC
- ▶ Équivalence des définitions : standard $\iff$ alternative
 - ▶ ($\Leftarrow$) répétition tant que ÉCHEC $\rightarrow$ résultat correct, espérance de complexité
 - ▶ ($\Rightarrow$) fixer une complexité maximale et renvoyer ÉCHEC si on dépasse cette complexité

Las Vegas vs. Monte Carlo

- ▶ Transformation d'un algorithme *Las Vegas* (alternatif) en algorithme *Monte Carlo*
 - ▶ remplacer ÉCHEC par une valeur arbitraire
- ▶ Transformation inverse pas toujours possible $\rightarrow$ nécessite de savoir *vérifier la solution*



idem Monte Carlo
pas de résultat faux

Bilan sur les algorithmes probabilistes

Pourquoi des algorithmes probabilistes ?

- ▶ Algorithmes souvent simples et efficaces
- ▶ Parfois : meilleure complexité que les algorithmes déterministes
- ▶ *Et pourquoi pas ?* en pratique, ils fonctionnent très bien !

Analyse des algorithmes probabilistes

- ▶ Modélisation probabiliste, avec variable(s) aléatoire(s)
- ▶ *Las Vegas* : étude de l'espérance de la complexité
- ▶ *Monte Carlo* : étude de la probabilité de succès

Et ensuite ?

- ▶ *Atlantic City* : « souvent correct, souvent rapide » (→ *Monte Carlo*)
- ▶ Algorithmes quantiques : généralisation des algorithmes probabilistes

Algorithmes probabilistes parfois difficiles à analyser... mais indispensables !

Ce que ne sont pas les algorithmes probabilistes

Algorithmes heuristiques

- ▶ Algo. heuristique : méthode non prouvée, qui fonctionne souvent en pratique
 - ▶ souvent : basée sur des choix aléatoires
 - ▶ parfois : fonctionne sur *presque toutes* les entrées
- ▶ Algo. probabiliste : méthode prouvée quelque soit l'entrée

Analyse en moyenne

- ▶ Analyse en moyenne :
 - ▶ comportement d'algorithmes sur des *entrées aléatoires*
 - ▶ espérance de la complexité sur une entrée aléatoire
 - ▶ question subtile : quelle distribution sur les entrées ?
- ▶ Analyse d'algo. probabiliste : espérance de la complexité quelque soit l'entrée
- ▶ Liens : certains algorithmes probabilistes reviennent à *rendre aléatoire* l'entrée

Ce cours : uniquement analyse d'algorithmes probabilistes

Table des matières

1. Qu'est-ce qu'un algorithme probabiliste ?

2. Exemple 1 : coupe minimale

3. Exemple 2 : tri rapide

Coupe minimale dans un graphe

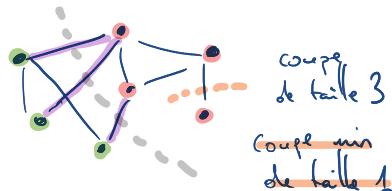
Définition

- ▶ **Coupe** dans un graphe $G = (S, A)$: partition de $S = S_1 \sqcup S_2$, avec $S_1, S_2 \neq \emptyset$
- ▶ Taille d'une coupe $S_1 \sqcup S_2$: nombre d'arêtes entre S_1 et S_2
$$= \#\{\{u_1, u_2\} \in A : u_1 \in S_1, u_2 \in S_2\}$$

Problème de la coupe minimale

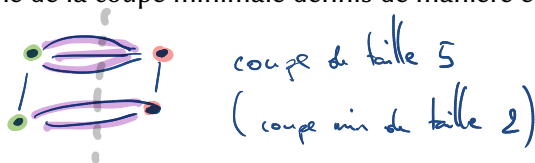
Entrée : Graphe $G = (S, A)$

Sortie : Coupe $S = S_1 \sqcup S_2$ de taille minimale



Généralisation nécessaire : multigraphes

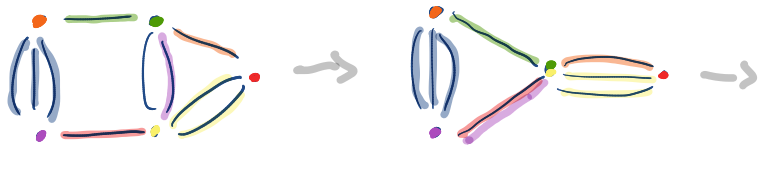
- ▶ Un *multigraphe* est un graphe qui autorise plusieurs arêtes entre 2 sommets
- ▶ Coupe et problème de la coupe minimale définis de manière équivalente



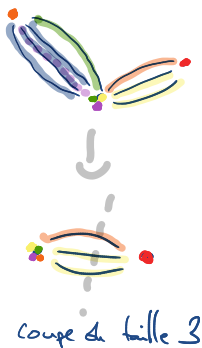
Algorithme probabiliste pour la coupe minimale

COUPEMIN(G):

1. Tant que G possède au moins 3 sommets :
2. Choisir une arête de G , uniformément
3. *Contracter* l'arête G
4. Renvoyer la coupe définie par les deux sommets restants



Remarque Coupe obtenue de taille 3
alors que la coupe min. est de taille 1.



Complexité de COUPEMIN

Fait admis

Si un multigraphe G à n sommets est représenté par listes d'adjacence, la *contraction* d'une arête peut s'effectuer en temps $O(n)$

Théorème

L'algorithme COUPEMIN renvoie une coupe de G en temps $O(n^2)$

Preuve

- La boucle Tant que est exécutée $\Theta(n)$ fois.
 - D'après le fait admis, chaque exécution de la boucle coûte $\Theta(n)$.
- $\Rightarrow$ Complexité $\Theta(n^2)$.

Correction de COUPEMIN

Pourquoi cet algorithme fonctionnerait-il ?

Lemme de correction

L'algorithme COUPEMIN appliqué à un multigraphe à n sommets renvoie une coupe *minimale* avec probabilité $\geq \frac{2}{n(n-1)}$

Remarque

- ▶ Cette probabilité est très petite !
- ▶ Exemple pour $n = 100$, $\frac{2}{n(n-1)} \simeq 0,02\%$...

Rappel En répétant N fois un algo qui a une proba de succès p ,
la proba d'être toujours incorrect est $\leq e^{-pN}$

Preuve du lemme de correction

- Pour mq l'algo renvoie une coupe minimale avec proba $\geq \frac{2}{n(n-1)}$, on va fixer une coupe minimale arbitraire C^* et mq l'algo renvoie cette coupe avec proba $\geq \frac{2}{n(n-1)}$ ($\Leftrightarrow$ aucune arête de C^* n'est contractée)
- Quand il reste k sommets, quelle est la proba de choisir une arête de C^* ?
 $\hookrightarrow \frac{\# \text{arêtes de } C^*}{\# \text{arêtes total}} \rightsquigarrow m : G \text{ a } \geq \frac{km}{2} \text{ arêtes}$

Preuve chaque sommet a au moins m voisins, donc il y a $\geq \frac{k \cdot m}{2}$ arêtes

 $\Rightarrow$ Proba de choisir une arête de C^* est $\leq \frac{m}{km/2} = 2/k$.
- Proba de ne jamais choisir d'arête de C^* :
$$\geq \left(1 - \frac{2}{n}\right) \left(1 - \frac{2}{n-1}\right) \cdots \left(1 - \frac{2}{3}\right) = \prod_{k=3}^n \frac{k-2}{k} = \frac{\prod_{k=3}^n (k-2)}{\prod_{k=3}^n k} = \frac{1 \cdot 2}{n(n-1)}$$

Répétitions de COUPEMIN

Probabilité de succès très faible $\rightarrow$ on répète l'algorithme pour améliorer cette probabilité

Lemme de répétition

Si on répète N fois COUPEMIN et qu'on garde la plus petite coupe renvoyée, cette coupe est minimale avec probabilité $\geq 1 - e^{-2N/n(n-1)}$

Exemple

Avec $N = 3n^2$ répétitions : complexité $O(n^4)$ et probabilité de succès $\geq 1 - e^{-6} > 99\%$

Preuve

Pour que la plus petite des coupes trouvées ne soit pas minimale, il faut qu'aucune ne soit minimale, donc qu'aucune répétition n'ait été un succès.

Bilan sur COUPEMIN

Complexité

- ▶ Un appel à COUPEMIN coûte toujours $O(n^2)$
- ▶ On a besoin de $O(n^2)$ répétitions $\rightarrow O(n^4)$

Correction

- ▶ Un appel à COUPEMIN renvoie une coupe minimale avec proba. $\geq \frac{2}{n(n-1)}$
- ▶ N appels à COUPEMIN renvoient une coupe minimale avec proba. $\geq 1 - e^{-2N/n(n-1)}$
- ▶ cn^2 appels à COUPEMIN renvoient une coupe minimale avec proba. $\geq 1 - e^{-2c}$

COUPEMIN répété est toujours en temps $O(n^4)$, et correct avec très bonne probabilité

- ▶ Type d'algorithme probabiliste: Monte Carlo

Table des matières

1. Qu'est-ce qu'un algorithme probabiliste ?

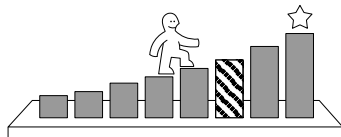
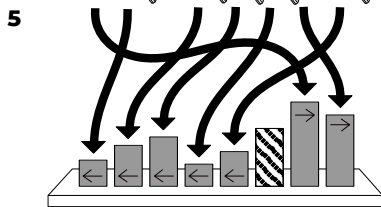
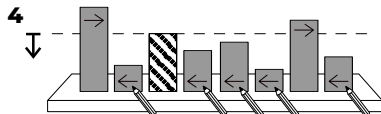
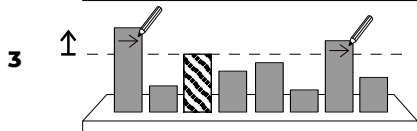
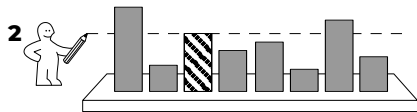
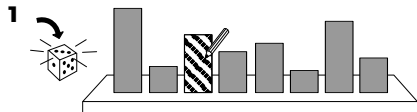
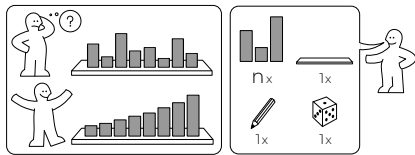
2. Exemple 1 : coupe minimale

3. Exemple 2 : tri rapide

KVICK SÖRT

idea-instructions.com/quick-sort/
v1.2, CC by-nc-sa 4.0

IDEA



Le tri rapide

TRI RAPIDE(T):

1. Si $\text{taille}(T) = 1$: renvoyer T
2. $p \leftarrow T[j]$ avec j choisi uniformément entre 0 et $n - 1$ (*pivot*)
3. $T^{\text{INF}}, T^{\text{SUP}} \leftarrow$ tableaux vides
4. Pour $i = 0$ à $n - 1$, $i \neq j$:
 5. Si $T[i] \leq p$: ajouter $T[i]$ à T^{INF}
 6. Sinon: ajouter $T[i]$ à T^{SUP}
7. $T^{\text{INF}} \leftarrow \text{TRI RAPIDE}(T^{\text{INF}})$
8. $T^{\text{SUP}} \leftarrow \text{TRI RAPIDE}(T^{\text{SUP}})$
9. Renvoyer la concaténation $T^{\text{INF}} \cdot [p] \cdot T^{\text{SUP}}$

Implantation

- Pas de tableaux annexes T^{INF} et T^{SUP}

- *Partition* de T en trois zones:

T^{INF}	p	T^{SUP}
------------------	-----	------------------

- Calcul des zones par échange de valeurs

cf. TP Programmation

Première analyse du tri rapide

Lemme (correction)

TRI RAPIDE(T) renvoie T trié

Preuve par récurrence

- Si $\#T = 1$ ou 0 : OK

- Sinon :
1. $T_{\text{INF}} \cup T_{\text{SUP}}$ contient tous les éléments
2. $\forall x \in T_{\text{INF}} \ y \in T_{\text{SUP}}, \ x < y$

3. les appels récursifs traitent T_{INF} et T_{SUP} .

Lemme (complexité)

Si les choix probabilistes sont *maximalement malchanceux*, le nombre de comparaisons effectuées est $O(n^2)$

Preuve

Choix le pire : pivot est le + petit (ou le + gros) élément de T

$$\hookrightarrow C_n = n - 1 + C_{n-1} = \dots = \Theta(n^2)$$

Espérance du nombre de comparaisons

Notations

- ▶ $T^{(i)}$: $i^{\text{ème}}$ plus petit élément de T
- ▶ $X_{ij} = \begin{cases} 1 & \text{si } T^{(i)} \text{ est comparé à } T^{(j)} \text{ au cours de l'algorithme} \\ 0 & \text{sinon} \end{cases}$
- ▶ X : nombre total de comparaisons $\rightarrow X = \sum_{i < j} X_{ij}$

Lemme

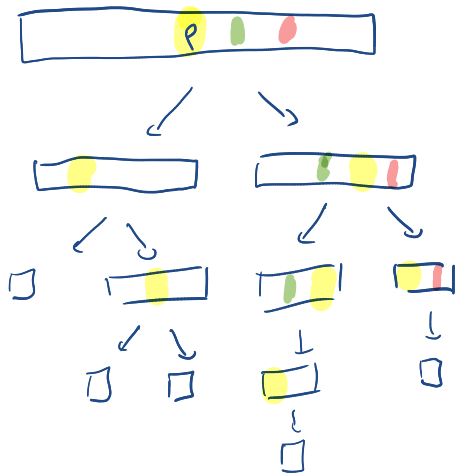
Pour $1 \leq i < j \leq n$, $\mathbb{E}[X_{ij}] = \Pr[X_{ij} = 1] = \frac{2}{j-i+1}$

Théorème

L'espérance du nombre de comparaisons effectuées par TRIRAPIDE est $O(n \log n)$

$$\begin{aligned} \mathbb{E}[X] &= \sum_{i < j} \mathbb{E}[X_{ij}] = \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} = 2 \cdot \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{1}{k} \leq 2 \sum_{i=1}^n \sum_{k=1}^n \frac{1}{k} \\ &\leq 2 \cdot \underline{n} \cdot \underline{\ln(n)} = O(n \log n) \end{aligned}$$

Preuve du lemme : $\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$



- $T^{(i)}$ est comparé à $T^{(j)}$ si, à un moment, l'un des deux est le pivot et l'autre est dans le même sous-tableau
- Ils ne sont jamais comparés si ils sont "séparés" par un pivot p .

$$\Rightarrow T^{(i)} < p < T^{(j)}$$

- On regarde le moment de leur séparation : le pivot p est choisi parmi $[T^{(i)}, T^{(i+1)}, \dots, T^{(j)}]$. Chaque élément de l'intervalle a proba $\frac{1}{j-i+1}$ d'être choisi. Donc $\Pr[p \in \{T^{(i)}, T^{(j)}\}] = \frac{2}{j-i+1}$.

Bilan sur le TRIRAPIDE

Propriétés du TRIRAPIDE

- ▶ L'algorithme est toujours correct
- ▶ L'espérance de sa complexité est $O(n \log n)$
- ▶ Type d'algorithme probabiliste: *Las Vegas*

Comportement pratique

- ▶ Le TRIRAPIDE est efficace en pratique, s'il est implanté *en place*
- ▶ Tri *stable*