Modèles de calcul, Complexité, Approximation et Heuristiques

Modèle probabiliste: Algorithmes et Complexité

Jean-Louis Roch

Master-2 Mathématique – Informatique

Grenoble-INP – UJF Grenoble University, France

Plan du cours

Plan du cours

1. **Introduction. Atlantic City, Monte-Carlo, Las Vegas. Exemple de arbre ET-OU : analyse et borne inférieure.**
2. Techniques d'analyse de coût et de construction d'algorithmes probabilistes. Applications : arithmétique ; recherche et tri ; graphes.
3. Classes de complexité probabilistes.
4. "Dérandomisation". Algorithmes interactifs, classes IP et PCP.

Bibliographie

- ***An introduction to randomized algorithms***, R.M. Karp, Discrete Applied Mathematics, 34 :165-201, 1991.

  Preprint : ICSI Technical Report TR-90-024,

  http://www.icsi.berkeley.edu/cgi-bin/pubs/publication.pl?ID=000588
- Randomized Algorithms. R. Motwani and P. Taghavan, Cambridge Universty Press, 1995
- M. Mitzenmacher and E. Upfal. Probability and Computing : Randomized Algorithms and Probabilistic Analysis. Cambridge University Press, New York (NY), 2005.
- Introduction à l'algorithmique probabiliste, Philippe Duchon (v4), 2007
- Computational Complexity: A Modern Approach. S. Arora, B. Barak. Cambrdge Univ. Press (2009) [1]raft available with chapter 7 Randomized computation and Chapter 8 Interactive proofs

1. D

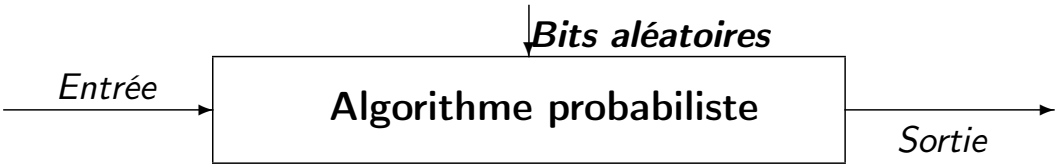
Algorithmes déterministes

Objectif

Quelle que soit l'entrée,
l'algorithme déterministe doit délivrer une sortie **correcte**
... et *rapidement* [a].

a. typiquement en temps polynomial du nombre de bits en entrée.

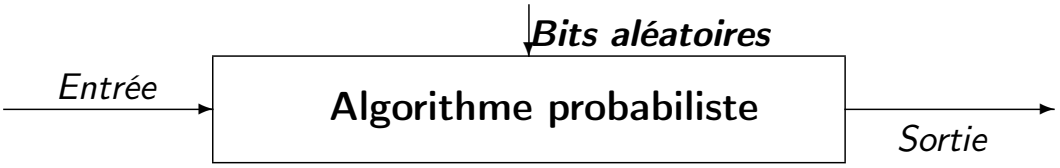
Algorithmes probabilistes



En plus de la donnée en entrée, un algorithme probabiliste prend une source de bits aléatoires :

- il peut l'utiliser pour faire des choix durant l'exécution ;
- le comportement peut varier même si la donnée en entrée est fixée.

Algorithmes probabilistes



En plus de la donnée en entrée, un algorithme probabiliste prend une source de bits aléatoires :

- il peut l'utiliser pour faire des choix durant l'exécution ;
- le comportement peut varier même si la donnée en entrée est fixée.

Objectif

Quelle que soit l'entrée,
l'algorithme probabiliste délivre une sortie probablement **correcte**.
La probabilité dépend uniquement des tirages effectués.

Ne pas confondre !

Ne pas confondre avec l'analyse probabiliste d'un algorithme déterministe (par exemple analyse en moyenne).

Cela est très différent d'un algorithme probabiliste :

- l'entrée est ici supposée suivre une distribution ;
- l'analyse prouve que l'algorithme donne un résultat correct pour la plupart des entrées.

Avantage des algorithmes probabilistes

- Simplicité
- Performance

Exemple 1.
Comparer un algorithme déterministe ou probabiliste pour le problème :

- Entrée : un tableau A de n booléens, dont au moins 75% valent VRAI.
- Sortie : i tel que $A[i]$ vaut VRAI.

Avantage des algorithmes probabilistes
- Simplicité
- Performance

Exemple 1.
Comparer un algorithme déterministe ou probabiliste pour le problème :
- Entrée : un tableau A de n booléens, dont au moins 75% valent VRAI.
- Sortie : i tel que $A[i]$ vaut VRAI.

Soit $p < 25\%$ la proportion de FAUX :

- Det :
  for i=1..n do { if A[i] return i ; }
- Probabiliste (LV) en temps borné = K essais :
  for t=1..K do { i= rand(1,n) ; if A[i] return i ; }
  return ECHEC ; // avec Probabilité d'échec = $\frac{1}{p^K} = \frac{1}{4^K}$,
  Pour $K = 5$, prob(échec) $< 10^{-3}$. Pour $k = 200$, prob(échec) $< 10^{-120}$.
- Probabiliste en temps non borné
  while(1) { i= rand() ; if A[i] return i ; }
  Temps moyen = $\sum_{k=1}^{\infty} kp^{k-1}(1-p) = (1-p)\sum_{k=1}^{\infty} kp^{k-1}$.
  Or
  $F(t) = \sum_{k=1}^{\infty} kt^{k-1} = \sum_{k=1}^{\infty} \frac{\mathrm{d}}{\mathrm{d}t} t^k = \frac{\mathrm{d}}{\mathrm{d}t} \sum_{k=1}^{\infty} t^k = \frac{\mathrm{d}}{\mathrm{d}t} \frac{1}{1-t} - 1 = \frac{1}{(1-t)^2}$.
  Donc Temps moyen = $\frac{1}{1-p} = \frac{4}{3}$.

Exemple 2 : multiplication avec table erronée

- On sait calculer correctement toute addition, mais pour les multiplications, on utilise T une table avec au moins 90% de données correctes.

	5	6	7	8	9
5	25	30	35	40	45
6	30	36	42	48	52
7	35	42	49	56	63
8	40	48	56	64	72
9	45	52	63	72	81

- Comment calculer correctement $a \times b$ (i.e. $T[a, b]$) ?

Exemple 2 : multiplication avec table erronée
- On sait calculer correctement toute addition, mais pour les multiplications, on utilise T une table avec au moins 90% de données correctes.

	5	6	7	8	9
5	25	30	35	40	45
6	30	36	42	48	52
7	35	42	49	56	63
8	40	48	56	64	72
9	45	52	63	72	81

- Comment calculer correctement $a \times b$ (i.e. $T[a, b]$) ?

Taux d'erreur maximal supporté dans la table : 25%

Exemple 2 : multiplication avec table erronée

- On sait calculer correctement toute addition, mais pour les multiplications, on utilise T une table avec au moins 90% de données correctes.

	5	6	7	8	9
5	25	30	35	40	45
6	30	36	42	48	52
7	35	42	49	56	63
8	40	48	56	64	72
9	45	52	63	72	81

- Comment calculer correctement $a \times b$ (i.e. $T[a, b]$) ?
- Algorithme probabiliste :
  Soit n_r (resp. n_c) le nombre de lignes (resp. colonnes) ;
  soit i_a l'indice ligne de a et i_b l'indice colonne de b.
  1. choisir au hasard $i_u \in \{0, \ldots, n_r - 1\}$ et $i_v \in \{0, \ldots, n_c - 1\}$;
  2. Lire $m_1 = T[i_a + i_u, i_b + i_v]$, $m_2 = T[i_a + i_u, i_v]$, $m_3 = T[i_u, i_b + i_v]$, $m_4 = T[i_u, i_v]$;
  3. retourner $r = m_1 - m_2 - m_3 + m_4$.

  // soit $(a + u)(b + v) - (a + u)v - u(b + v) + uv$, je ab

Exemple 2 : multiplication avec table erronée

- On sait calculer correctement toute addition, mais pour les multiplications, on utilise T une table avec au moins 90% de données correctes.

	5	6	7	8	9
5	25	30	35	40	45
6	30	36	42	48	52
7	35	42	49	56	63
8	40	48	56	64	72
9	45	52	63	72	81

- Comment calculer correctement $a \times b$ (i.e. $T[a, b]$) ?
- Algorithme probabiliste :
  Soit n_r (resp. n_c) le nombre de lignes (resp. colonnes) ;
  soit i_a l'indice ligne de a et i_b l'indice colonne de b.
  1. choisir au hasard $i_u \in \{0, \ldots, n_r - 1\}$ et $i_v \in \{0, \ldots, n_c - 1\}$;
  2. Lire $m_1 = T[i_a + i_u, i_b + i_v]$, $m_2 = T[i_a + i_u, i_v]$,
     $m_3 = T[i_u, i_b + i_v]$, $m_4 = T[i_u, i_v]$;
  3. retourner $r = m_1 - m_2 - m_3 + m_4$.
     // soit $(a + u)(b + v) - (a + u)v - u(b + v) + uv$, ie ab

Taux d'erreur maximal supporté dans la table : 25%

Exemple 2 : multiplication avec table erronée

- Probabilité erreur : $p = \Pr(r \neq a \times b) \leq \sum_{i=1}^{4} \Pr(m_i \text{ erroné}) \leq 0.4$.
  La probabilité p que le résultat soit correct est supérieure à $0,6 > \frac{1}{2}$.

Exemple 2 : multiplication avec table erronée
Probabilité erreur : $p = \Pr(r \neq a \times b) \leq \sum_{i=1}^{4} \Pr(m_i \text{ erroné}) \leq 0.4$.
La probabilité p que le résultat soit correct est supérieure à $0,6 > \frac{1}{2}$.

Attention : $\Pr(m_i \text{ correct}) \geq p$, mais comme les evts ne sont pas indépendants, on n'a pas $\Pr(r = a \times b) = \prod_{i=1}^{4} \Pr(m_i \text{ correct})$.

Exemple 2 : multiplication avec table erronée

- Si on exécute cet algorithme n fois, et que l'on retourne la valeur qui apparait plus de $n/2$ fois, la probabilité d'erreur ϵ tend rapidement vers 0.
  - $\epsilon = \Pr(\#\text{corrects} < n/2) = \sum_{i=0}^{n/2} C_n^i (1-p)^i p^{n-i} = F(k, n, p)$
    cumulative distribution function
  - Via l'inégalité de Hoeffding, pour $p < 0.4$, on a :
    $\epsilon = \Pr(\#\text{corrects} < n/2) \leq \min\left(2e^{-0.02n}; \frac{3}{n}\right)$.

Si on exécute cet algorithme n fois, et que l'on retourne la valeur qui apparait plus de $n/2$ fois, la probabilité d'erreur ϵ tend rapidement vers 0.
- $\epsilon = \Pr(\#\text{corrects} < n/2) = \sum_{i=0}^{n/2} C_n^i (1-p)^i p^{n-i} = F(k,n,p)$ *cumulative distribution function*
- Via l'inégalité de Hoeffding, pour $p < 0.4$, on a : $\epsilon = \Pr(\#\text{corrects} < n/2) \leq \min\left(2e^{-0.02n}, \frac{2}{n}\right)$.

Attention : $\Pr(m_i \text{ correct}) \geq p$, mais comme les evts ne sont pas indépendants, on n'a pas $\Pr(r = a \times b) = \prod_{i=1}^{4} \Pr(m_i \text{ correct})$.

Résultat majoritaire d'exécutions indépendantes

- On fait n exécutions indépendantes avec la même entrée d'un algorithme probabiliste de probabilité d'erreur $p < 0.5$. Quelle est la probabilité ϵ qu'un même résultat qui apparaît $n/2$ fois soit erroné ?
  $\epsilon = \Pr(\#\text{corrects} < n/2) = \sum_{i=0}^{n/2} C_n^i (1-p)^i p^{n-i}$ *(cumulative distribution fn)*
- Majoration naïve par l'inégalité de Markov :
  $\Pr(\text{erreur}) \leq \Pr(\#\text{faux} \geq n/2) \leq \frac{E[\#\text{faux}]}{n/2} = 2p$
- Mieux : majoration de l'écart à la moyenne d'une somme de va Bernoulli, en utilisant les inégalités de Hoeffding et Bienaymé-Tchebychev :

$$\Pr(\text{erreur}) \leq \min\left\{2\exp\left(-2\left(\frac{1}{2}-p\right)^2 n\right) ; \quad \frac{2p(1-p)}{1-2p}.n^{-1}\right\}.$$

- Application : table de multiplication avec 10% d'erreurs :
  $p = 0.4 \Longrightarrow \Pr(\text{erreur}) \leq \min\left\{2e^{-0.02n}; 2.4n^{-1}\right\}$.

Détails de la majoration de l'erreur par inégalités d'Hoeffding et de Bienaymé-Tchebychev

- majoration de l'écart à la moyenne d'une somme de va Bernoulli X_k indépendantes avec $\Pr(X_k = 1) = 1 - \Pr(X_k = 0) = p$.
  Soit $S_n = \sum_{k=1}^{n} X_k$, alors
  - $\Pr\left(|S_n - E[S_n]| \geq x\sqrt{n}\right) \leq 2\exp\left(-2x^2\right)$ (par inégalité de Hoeffding)
  - $\Pr\left(|S_n - E[S_n]| \geq x\sqrt{n}\right) \leq \frac{p(1-p)}{x^2}$ (par Bienaymé-Tchebychev)
- Application : posons $X_k = 1$ ssi la $k^{\text{ième}}$ exécution est erronée et $S_n = \sum_{k=1}^{n} X_k$:

  $$\Pr(\text{erreur}) \leq \Pr\left(S_n \geq n/2\right) = \Pr\left(S_n - E[S_n] \geq n/2 - E[S_n]\right).$$

  Or $E[S_n] = p.n$; en posant $x = \left(\frac{1}{2} - p\right)\sqrt{n}$:
  - l'inégalité d'Hoeffding donne :
    $\Pr(\text{erreur}) \leq \Pr\left(S_n - E[S_n] \geq x\sqrt{n}\right) \leq 2e^{-2x^2} = 2\sqrt{e^{(1-2p)^2 n}}$.
  - l'inégalité de Bienaymé-Tchebychev donne :
    $\Pr(\text{erreur}) \leq \Pr\left(S_n - E[S_n] \geq x\sqrt{n}\right) \leq \frac{p(1-p)}{x^2} = \frac{8p(1-p)}{(1-2p)^2}.n^{-1}$

Las Vegas, Monte-Carlo

Un algorithme probabiliste s'exécute pendant un temps fini et produit une réponse correcte avec une probabilité $\geq c > \frac{1}{2}$.

- Un algorithme Monte-Carlo peut toujours renvoyer une sortie incorrecte (avec une probabilité $\leq \frac{1}{2}$).
- Un algorithme Las Vegas renvoie soit une sortie correcte, soit une indication d'ECHEC (avec une probabilité $\leq \frac{1}{2}$).
  $\hookrightarrow$ *ne se trompe jamais, seul le temps d'exécution est une variable aléatoire de moyenne bornée (par exemple polynomiale).*

Las Vegas, Monte-Carlo, Atlantic City

Problèmes de décision - Sortie OUI / NON.

- Las Vegas : la sortie est soit correcte soit ECHEC
- Monte Carlo : la sortie peut être incorrecte. Deux cas :
  — erreur possible que la sortie soit OUI ou NON ;
  — erreur d'un seul côté (et dissymétrie OUI/NON)

Calculer $f(x) \in \{0,1\}$:

- Atlantic City : retourne y et $\Pr(f(x) = y) \geq c > 0.5$.
  $\hookrightarrow$ Both-sided error, classe de complexité BPP Bounded error Probability Polynomial.
- Monte-Carlo : si $f(x) = 1$, retourne 1 avec probabilité ≥ 0.5 ; si $f(x) = 0$, retourne 0.
  $\hookrightarrow$ One-sided error, classe de complexité RP Random Polynomial.
- Las Vegas : retourne $f(x)$ avec probabilité ≥ 0.5 ; ou sinon ECHEC avec probabilité < 0.5

Soit x une entrée du problème de décision f.

Atlantic City

	Sortie $A(x)$ de l'algorithme Atlantic City	
	NON	OUI
cas $f(x)$ =NON	prob $\geq c_1 > 0.5$	prob $\leq 1 - c_1$
cas $f(x)$ =OUI	prob $\leq 1 - c_2$	prob $\geq c_2 > 0.5$

Monte Carlo

	Sortie $A(x)$ de l'algorithme Monte-Carlo	
	NON	OUI
cas $f(x)$ =NON	prob $= 1$	
cas $f(x)$ =OUI	prob ≤ 0.5	prob ≥ 0.5

Las Vegas

	Sortie $A(x)$ de l'algorithme Las Vegas		
	NON	OUI	Echec
cas $f(x)$ =NON	prob ≥ 0.5		prob ≤ 0.5
cas $f(x)$ =OUI		prob ≥ 0.5	prob ≤ 0.5

Exemple de décision

Algorithme de Miller-Rabin

- Entrée : $n > 5$ un entier impair.
- Sortie : "PREMIER" ou "COMPOSE"
  - Si n est premier, renovoit toujours "PREMIER" ;
  - Si n est composé, renovoit au moins 3 fois sur 4 : "COMPOSE" (et sinon "PREMIER").

$\hookrightarrow$ Algorithme Monte-Carlo [de test de composition]

	Sortie de MillerRabin$A(n)$	
	PREMIER	COMPOSE
cas $f(x)$ =PREMIER (ie non composé)	prob $= 1$	
cas $f(x)$ =COMPOSE	prob ≤ 0.25	prob ≥ 0.75

Exemple 1 : multiplication avec table erronée

- Soit T une table de multiplication avec au moins 80% de données correctes.

	6	7	8	9
6	36	42	48	52
7	42	49	47	63
8	48	56	64	72
9	52	63	72	81

- Comment calculer correctement $a \times b$, pour a et b fixés (arbitraires) ?

Algorithme Las Vegas sans échec

Suppression de la sortie ECHEC

- Soit `LasVegas` un algorithme Las Vegas de probabilité d'échec p et de coût $T(x)$;
- `do {res := LasVegas(x); } while (LasVegas(x) == ECHEC)` est un algorithme probabiliste qui donne toujours une sortie correcte. Son coût moyen$= \frac{1}{1-p} T(x)$; son coût médian$= \frac{1}{\log_2(1/p)} T(x)$.

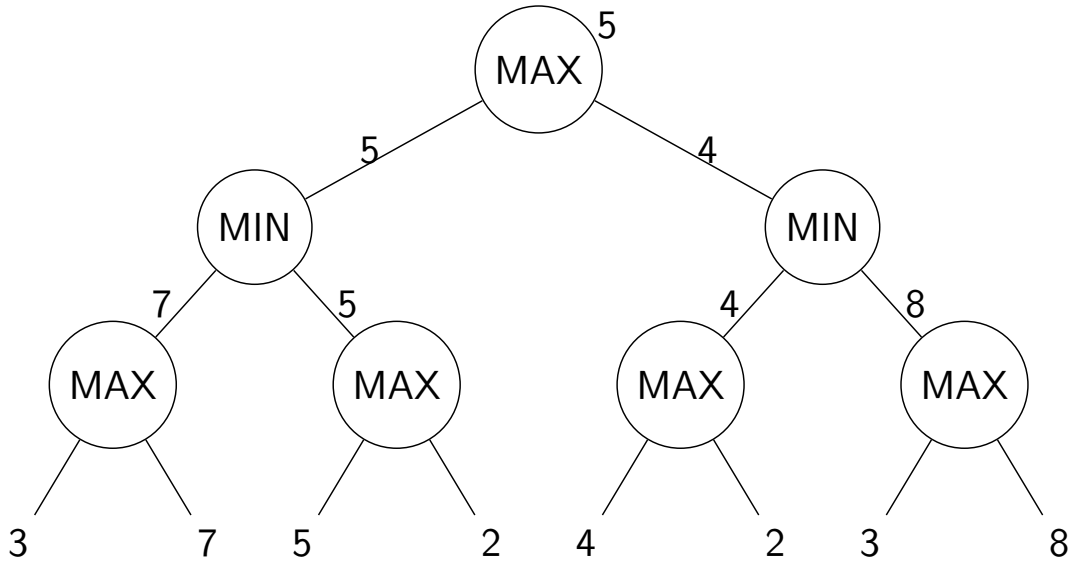
... et réciproquement

- Si il existe un algorithme probabiliste sans échec de coût *médian* $T(x)$
- alors il existe un algorithme Las Vegas de coût $T(x)$ et de probabilité d'échec $\frac{1}{2}$.

Quelques applications des algorithmes probabilistes

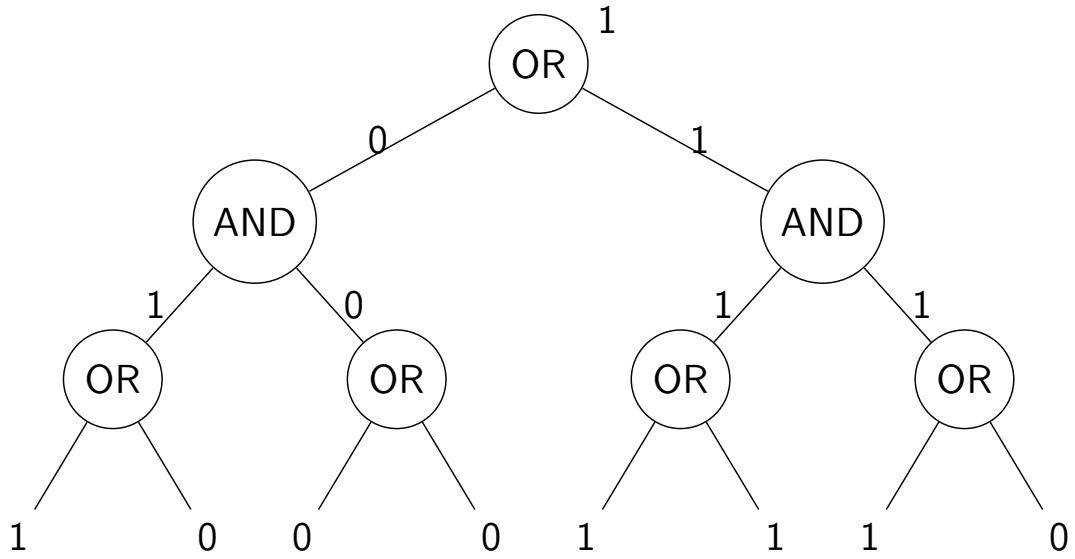
- Théorie des nombres
- Algèbre
- Programmation linéaire
- Tri, recherche, sélection
- Géométrie algorithmique
- Graphes : plus courts chemins, arbres couvrants, coupes minimales
- Dénombrement et structures combinatoires
- Calcul parallèle et distribué (consensus, routage, ordonnancement, ...)
- Preuves probabilistes (complexité interactive)
- Élimination de l'aléatoire pour construire un algorithme déterministe (*derandomization*)

Jeu à 2 joueurs – Évaluation arbre min-max



- Entrée : Arbre avec alternance de nœuds MIN et MAX ; chaque feuille a une valeur entière (évaluation).
- Sortie : la valeur de la racine.

Jeu à 2 joueurs – Évaluation arbre ET-OU



- Feuille à valeur booléenne, Min=ET, Max=OU.
- Arbre binaire complet et dense : $n = 2^h$ feuilles de hauteur h.
- Tout algorithme déterministe a un coût en pire cas $> n$.

Jeu à 2 joueurs – Évaluation arbre ET-OU

Algorithme probabiliste

- Évaluer (récursivement) un fils choisi au hasard
- Si le nœud n'est pas évalué, alors évaluer l'autre fils.

↪ Las Vegas de Prob(ECHEC) = 0 !

Analyse du coût

- Notation : $n = 2^h$ et $k = h/2$. Sur chaque branche de l'arbre il y a k nœuds OU et $h - k$ nœuds ET.
- **Lemme :** *pour un nœud ET (resp. OU) de hauteur $2i$ (resp. $2i + 1$), le cout moyen d'évaluation est $\leq 3^i$.*
  - Preuve par récurrence sur la hauteur.
- **Théorème :** *Le coût moyen de l'algorithme probabiliste est $O\left(n^{0.793}\right)$.*
  - Preuve : la racine AND est de hauteur $h = 2k + 1$, donc : coût $\leq 2k = 2h/2 = \sqrt{2}^{\log_2 n} = n^{\log_2 \sqrt{3}} = n^{0.793}$

Introduction.
Las Vegas, Monte-Carlo, Atlantic City.

Jeu à 2 joueurs – Évaluation arbre ET-OU

Preuve lemme : pour un nœud ET de hauteur $2i$, on a : cout $\leq 3^i$.

- Hypothèse sur un nœud AND de hauteur $2i$: coût $\leq 3^i$.
  Vraie pour i=0.
- Nœud OR de hauteur $2i + 1$:
  - Si valeur OR=0, les deux fils AND de hauteur $2i$ sont tirés et évalués à 0 : Cout moyen $\leq 2 \cdot 3^i$.
  - Si valeur OR=1, au moins l'un des au plus deux fils AND tirés s'évalue à 1 et termine l'évaluation :
    Cout moyen $\leq \frac{1}{2} \cdot 3^i + \frac{1}{2} \cdot 2 \cdot 3^i = \frac{3}{2} \cdot 3^i$.
- Nœud AND de hauteur $2i + 2$:
  - Si valeur AND=1, les 2 fils OR de hauteur $2i + 1$ sont tirés et doivent s'évaluer à 1 : Cout moyen $\leq 2 \cdot \frac{3}{2} \cdot 3^i = 3^{i+1}$.
  - Si valeur AND=0, au moins l'un des deux fils OR tirés s'évalue à 0 et termine l'évaluation :
    Cout $\leq \frac{1}{2} \cdot 2.3^i + \frac{1}{2}\left(\frac{3}{2} \cdot 3^i + 2 \cdot 3^i\right) = \frac{11}{4} \cdot 3^i \leq 3^{i+1}$.

Borne inférieure d'un algorithme probabiliste

Coût [en pire cas] d'un algorithme probabiliste

Soit A un algorithme probabiliste et I un ensemble d'instances en entrée et R la v.a. correspondant aux tirages effectués :

$$\text{Coût}(A, I) = \max_{i \in I} \text{Coût moyen}(A, i) = \max_{i \in I} E_R[\text{Coût}(A, i, R)]$$

Minorant [borne inférieure] de complexité probabiliste

Tout minorant [atteint] de $\min_{A \in \mathcal{R}} \max_{i \in I} E_R[\text{Coût}(A, i, R)]$
où $\mathcal{R}$ désigne l'ensemble des algorithmes probabilistes ;

Borne inférieure d'un algorithme probabiliste

Modélisation d'un algorithme probabiliste

Un algorithme probabiliste qui donne toujours une sortie correcte (exemple ET-OU précédent) peut être vu comme un algorithme qui tire au hasard, à chaque exécution, un algorithme déterministe résolvant le problème.

- Soit $\mathcal{A}_{Rand} = \{$ algorithmes probabilistes pour le problème$\}$; soit $\mathcal{A}_{Det} = \{$ algorithmes déterministes pour le problème$\}$; [NB les tirages aléatoires dans $\{0, 1\}$ impliquent des probabilités dans $\mathbb{Q}$].
- Un algorithme probabiliste $A \in \mathcal{A}_{Rand}$ est alors une variable aléatoire X_A à valeurs dans $\mathcal{A}_{Det}$;
- sur une instance $i \in I$, le coût moyen de $A(i)$ ne dépend que de la loi de probabilité de X_A :

$$E_R[\text{Coût}(A, i, R)] = E_{X_A}[\text{Coût}(X_A, i)].$$

Borne inférieure d'un algorithme probabiliste

Principe Minimax [Yao]

Soient X et Y deux v.a. indépendantes à valeurs dans S_X et S_Y et soit $f : S_X \times S_Y \to \mathbb{R}$. Alors :

$$\min_{a \in S_X} E[f(a, Y)] \leq \max_{b \in S_Y} E[f(X, b)].$$

Preuve : $\min_{a \in S_X} E[f(a, Y)] \leq E_X[E_Y[f(X, Y)]] \overset{[Fubini]}{=} E_Y[E_X[f(X, Y)]] \leq \max_{b \in S_Y} E[f(X, b)]$.

Application du principe Minimax au calcul de minorant

Pour tout algorithme probabiliste $A \in \mathcal{A}_{Rand}$ (v.a. $X_A \in \mathcal{A}_{Det}$) et toute distribution d'instance en entrée (v.a. $Y_I \in I$) :

$$\begin{aligned} \max_{i \in I} E_R[\text{Coût}(A, i, R)] &= \max_{i \in I} E_{X_A}[\text{Coût}(X_A, i)] \\ &\geq \min_{B \in \mathcal{A}_{Det}} E_{Y_I}[Cout(B, Y_I)]. \end{aligned}$$

- $\hookrightarrow$ minorant de complexité déterministe en moyenne
- sur une distribution d'entrée Y_I que l'on peut fixer arbitrairement

Exemple : borne inférieure pour arbre ET-OU (1/2)

Remarque : réduction à un arbre binaire NOR

Dans un arbre ET-OU de racine ET et de feuilles OU, le remplacement de tous les ET et OU par des NOR donne un arbre de même valeur.

Preuve : $((a \text{ OR } b) \text{ AND } (c \text{ OR } d)) \equiv ((a \text{ NOR } b) \text{ NOR } (c \text{ NOR } d)))$

Choix de la distribution d'entrée : v.a. Y_I

- Pour la v.a. Y_I, on choisit pour Y_I de tirer une valeur 1 pour chaque feuille (indépendante) avec probabilité p ;
- Pour simplifier, on choisit en plus de fixer p pour que tout NOR de même hauteur ait indépendamment la même probabilité de valoir 1

  $\Longrightarrow p = \frac{3-\sqrt{5}}{2}$

  car $p = \Pr[NOR(a, b) = 1] = \Pr[a = 0].\Pr[b = 0] = (1-p)^2 \iff p = \frac{3-\sqrt{5}}{2}$.

Exemple : Minorant de complexité pour arbre ET-OU (2/2)

Minorant d'un algorithme déterministe pour la distribution choisie

- Soit $a \in \mathcal{A}_{Det}$ un algorithme déterministe ;
- soit $W_a(h)$ le nombre moyen (pour Y_I) de feuilles inspectées pour évaluer un nœud de hauteur h :

$$E_{Y_I}[\text{Cout}(a, Y_I)] \geq W_a(h).$$

- On minore le coût d'un algorithme déterministe en évaluant tout un sous-arbre d'un nœud avant d'inspecter une feuille de l'autre sous-arbre :

$$W_a(h) \geq W_a(h-1) + (1-p)W_a(h-1) = (2-p)W_a(h-1).$$

D'où $W_a(h) \geq (2-p)^h \left(\frac{1+\sqrt{5}}{2}\right)^h = n^{\log_2 \frac{1+\sqrt{5}}{2}} = \Omega\left(n^{0.694}\right)$.
Et donc $\min_{a \in \mathcal{A}_{Det}} E[\text{Cout}(a, Y_I)] = \Omega\left(n^{0.694}\right)$.

De ninorant à borne inférieure pour arbre ET-OU

Comment obtenir un meilleur minorant ?

- Si les deux fils d'un NOR valent 1 : aucun "bon" algorithme n'évalue les deux fils pour donner la réponse 0.
  $\hookrightarrow$ choix d'une distribution qui exclut ce cas.
- Après calcul [Saks&Wigderson86], on obtient le minorant $n^{\log_2 3/2} = n^{0,792}$

Theorem

L'algorithme probabiliste d'évaluation proposé est optimal parmi tous les algorithmes probabilistes.

Ce qui a été abordé aujourd'hui...

- Exemple d'algorithme probabiliste (calcul avec données erronnées)
- Atlantic City, Monte Carlo, Las Vegas
  de Las Vegas à probabiliste sans échec (en moyenne)
- Gain de complexité en pire cas : arbre ET-OU