Probabilités et Simulation

Générateurs de loi uniforme et de lois discrètes

Jean-Marc Vincent [1]

[1] Laboratoire d'Informatique de Grenoble
Polytech Grenoble

Octobre 2015

Outline

Histoires de dés

Pièces, dés, roues,...

Mécanisme physique :

Suite d'observations : $x_1, x_2, x_3, \cdots, x_n, \cdots$ à valeur dans $\{1, 2, \cdots, K\}$

Modèle probabiliste :

La séquence d'observations est modélisée par une suite de

- variables aléatoires,
- indépendantes,
- identiquement distribuées,
- de loi uniforme sur l'ensemble $\{1, 2, \cdots, K\}$ notée $\{X_n\}_{n \in \mathbb{N}}$

Notations et propriétés

Pour tout n et pour toute séquence $\{x_1, \cdots, x_n\}$ de $\{1, 2, \cdots, K\}^n$

$$\begin{aligned} \mathbb{P}(X_1 = x_1, \cdots, X_n = x_n) &= \mathbb{P}(X_1 = x_1). \cdots .\mathbb{P}(X_n = x_n) \text{ indépendance;} \\ &= \mathbb{P}(X = x_1). \cdots .\mathbb{P}(X = x_n) \text{ même loi;} \\ &= \frac{1}{K} \cdots \frac{1}{K} = \frac{1}{K^n} \text{ loi uniforme.} \end{aligned}$$

Outline

Histoires de dés (suite)

Pièce $\mapsto$ Dé-8

À partir d'une pièce de monnaie écrire un générateur aléatoire d'un dé à 8 faces:

Histoires de dés (suite)

Pièce $\mapsto$ Dé-8

À partir d'une pièce de monnaie écrire un générateur aléatoire d'un dé à 8 faces:

Dé-8()

Données: Une fonction "Pièce()" générateur aléatoire de $\{0, 1\}$
Résultat: Une séquence i.i.d. de loi uniforme sur $\{1, \cdots, 8\}$

$A_0 =$Pièce()
$A_1 =$Pièce()
$A_2 =$Pièce()
$S = A_0 + 2 * A_1 + 4 * A_2 + 1$
return S

Histoires de dés : Preuve de l'algorithme

Spécification :
une séquence d'appels à la fonction **Dé-8()** est modélisée par une séquence de variables aléatoires i.i.d. de loi uniforme sur $\{1, \cdots, 8\}$.

Histoires de dés : Preuve de l'algorithme

Spécification :
une séquence d'appels à la fonction **Dé-8()** est modélisée par une séquence de variables aléatoires i.i.d. de loi uniforme sur $\{1, \cdots, 8\}$.

Hypothèse :
$P_0, P_1, \cdots, P_n, \cdots$ séquence des appels à **Pièce()** iid de loi uniforme sur $\{0, 1\}$

Histoires de dés : Preuve de l'algorithme

Spécification :
une séquence d'appels à la fonction **Dé-8()** est modélisée par une séquence de variables aléatoires i.i.d. de loi uniforme sur $\{1, \cdots, 8\}$.

Hypothèse :
$P_0, P_1, \cdots, P_n, \cdots$ séquence des appels à **Pièce()** iid de loi uniforme sur $\{0, 1\}$

Preuve :
On note $S_0, S_1, \cdots, S_n, \cdots$ la séquence de variables aléatoires modélisant les résultats obtenus par appels successifs de **Dé-8()**.
Soit $n \in \mathbb{N}$ et $(x_0, x_1, \cdots, x_n) \in \{1, \cdots, 8\}^n$. Il faut montrer que

$$\mathbb{P}(S_0 = x_0, \cdots, S_n = x_n) = \frac{1}{8^{n+1}} \qquad \text{cqfd.}$$

Histoires de dés :Preuve de l'algorithme (suite)

On a

$$\mathbb{P}(S_0 = x_0, \cdots, S_n = x_n)$$

$$= \mathbb{P}(S_0 = x_0) \cdots \mathbb{P}(S_n = x_n)$$

car S_k ne dépend que de $P_{3k}, P_{3k+1}, P_{3k+2}$ et que les P_i sont indépendar

les $S_0, \cdots, S_n, \cdots$ sont donc indépendants;

$$= \mathbb{P}(S_0 = x_0) \cdots \mathbb{P}(S_0 = x_n) \text{ car } (P_{3k}, P_{3k+1}, P_{3k+2}) \text{ ont même loi.}$$

Histoires de dés :Preuve de l'algorithme (suite)

On a

$$\begin{aligned}
&\mathbb{P}(S_0 = x_0, \cdots, S_n = x_n) \\
&= \mathbb{P}(S_0 = x_0) \cdots \mathbb{P}(S_n = x_n) \\
&\quad \text{car } S_k \text{ ne dépend que de } P_{3k}, P_{3k+1}, P_{3k+2} \text{ et que les } P_i \text{ sont indépendar} \\
&\quad \text{les } S_0, \cdots, S_n, \cdots \text{ sont donc indépendants;} \\
&= \mathbb{P}(S_0 = x_0) \cdots \mathbb{P}(S_0 = x_n) \text{ car } (P_{3k}, P_{3k+1}, P_{3k+2}) \text{ ont même loi.}
\end{aligned}$$

Or pour i dans $\{1, \cdots, 8\}$, $i - 1$ s'écrit de manière unique en binaire $i - 1 =_2 a_2 a_1 a_0$.

$$\begin{aligned}
\mathbb{P}(S_0 = i) &= \mathbb{P}(P_0 = a_0, P_1 = a_1, P_2 = a_2) \\
&= \mathbb{P}(P_0 = a_0)\mathbb{P}(P_1 = a_1)\mathbb{P}(P_2 = a_2) \text{ les appels à Piece() sont indé} \\
&= \frac{1}{2}\frac{1}{2}\frac{1}{2} = \frac{1}{8} \text{ car même loi uniforme sur } \{0, 1\}.
\end{aligned}$$

d'où

$$\mathbb{P}(S_0 = x_0, \cdots, S_n = x_n) = \frac{1}{8^{n+1}} \qquad \text{cqfd.}$$

Histoires de dés (suite)

Pièce $\mapsto$ Dé-2^k

À partir d'une pièce de monnaie écrire un générateur aléatoire d'un dé à 2^k faces:

Histoires de dés (suite)

Pièce $\mapsto$ Dé-2^k

À partir d'une pièce de monnaie écrire un générateur aléatoire d'un dé à 2^k faces:

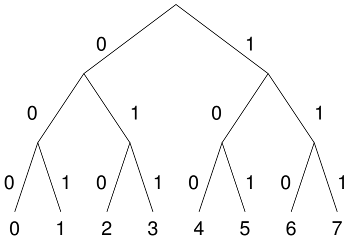
Dé(k)

Données: Une fonction "Pièce()" générateur aléatoire de $\{0, 1\}$
Résultat: Une séquence i.i.d. de loi uniforme sur $\{1, \cdots, 2^k\}$

S=0
for $i = 1$ **to** k
  S=Pièce() +2.S `// cf Schéma de Hörner`
$S = S + 1$
return S

Preuve: Identique au **Dé-8**, unicité de la décomposition binaire d'un entier de $\{0, \cdots, 2^k - 1\}$ par un vecteur de k bits.

Représentation binaire :



$$5 =_2 101, \; 2 =_2 010, \; 42 =_2 101010 \cdots$$

Histoires de dés (suite)

Pièce $\mapsto$ Dé-6

À partir d'une pièce de monnaie écrire un générateur aléatoire d'un dé à 6 faces:

Histoires de dés (suite)

Pièce $\mapsto$ Dé-6

À partir d'une pièce de monnaie écrire un générateur aléatoire d'un dé à 6 faces:

Dé-6()

Données: Une fonction **Dé-8()** générateur aléatoire de $\{1, \cdots, 8\}$
Résultat: Une séquence i.i.d. de loi uniforme sur $\{1, \cdots, 6\}$

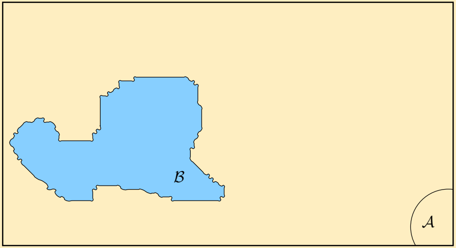
repeat
| X =**Dé-8()**
until $X \leqslant 6$
return X

Preuve: voir plus tard

Méthode basée sur le rejet

Principe

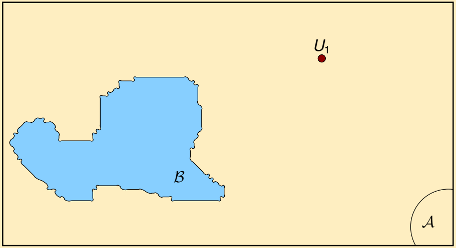
Générer uniformément sur $\mathcal{A}$ accepter si le point est dans $\mathcal{B}$.



Méthode basée sur le rejet

Principe

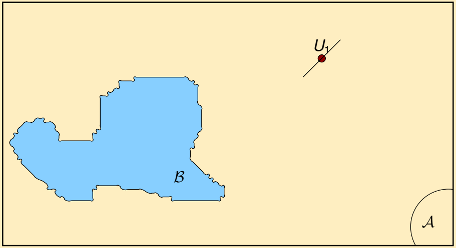
Générer uniformément sur $\mathcal{A}$ accepter si le point est dans $\mathcal{B}$.



Méthode basée sur le rejet

Principe

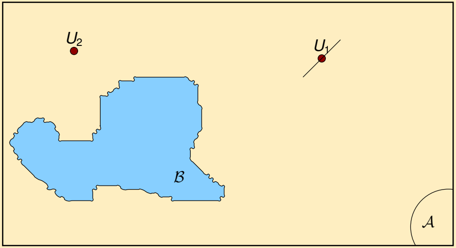
Générer uniformément sur $\mathcal{A}$ accepter si le point est dans $\mathcal{B}$.



Méthode basée sur le rejet

Principe

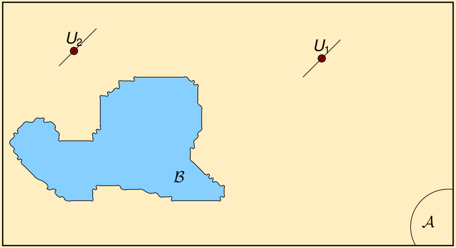
Générer uniformément sur $\mathcal{A}$ accepter si le point est dans $\mathcal{B}$.



Méthode basée sur le rejet

Principe

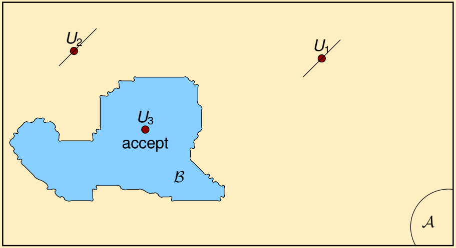
Générer uniformément sur $\mathcal{A}$ accepter si le point est dans $\mathcal{B}$.



Méthode basée sur le rejet

Principe

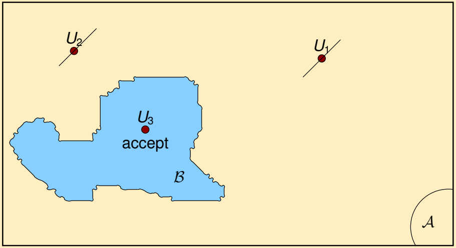
Générer uniformément sur $\mathcal{A}$ accepter si le point est dans $\mathcal{B}$.



Méthode basée sur le rejet

Principe

Générer uniformément sur $\mathcal{A}$ accepter si le point est dans $\mathcal{B}$.



Algorithme

Génère-unif($\mathcal{B}$)

Données:
Générateur uniforme sur $\mathcal{A}$
Résultat:
Générateur uniforme sur $\mathcal{B}$

repeat
  X =**Génère-unif($\mathcal{A}$)**
until $X \in \mathcal{B}$
return X

Méthode basée sur le rejet : preuve

Génère-unif($\mathcal{B}$)

Données:
Générateur uniforme sur $\mathcal{A}$
Résultat:
Générateur uniforme sur $\mathcal{B}$

$N = 0$
repeat
  X =**Génère-unif($\mathcal{A}$)**
  $N = N + 1$
until $X \in \mathcal{B}$
return X, N

Preuve

Tirages **Génère-unif($\mathcal{A}$)**: $X_1, X_2, \cdots, X_n, \cdots$

$$\begin{aligned}
& \mathbb{P}(X \in \mathcal{C}, N = k) \\
= \; & \mathbb{P}(X_1 \notin \mathcal{B}, \cdots, X_{k-1} \notin \mathcal{B}, X_k \in \mathcal{C}) \\
= \; & \mathbb{P}(X_1 \notin \mathcal{B}) \cdots \mathbb{P}(X_{k-1} \notin \mathcal{B}) \mathbb{P}(X_k \in \mathcal{C}) \\
= \; & \left(1 - \frac{|\mathcal{B}|}{|\mathcal{A}|}\right)^{k-1} \frac{|\mathcal{C}|}{|\mathcal{A}|}
\end{aligned}$$

$$\begin{aligned}
\mathbb{P}(X \in \mathcal{C}) &= \sum_{k=1}^{+\infty} \mathbb{P}(X \in \mathcal{C}, N = k) \\
&= \sum_{k=1}^{+\infty} \left(1 - \frac{|\mathcal{B}|}{|\mathcal{A}|}\right)^{k-1} \frac{|\mathcal{C}|}{|\mathcal{A}|} = \frac{|\mathcal{C}|}{|\mathcal{B}|}
\end{aligned}$$

Donc la loi est **uniforme** sur $\mathcal{B}$

Méthode basée sur le rejet : complexité

Génère-unif($\mathcal{B}$)

Données:
Générateur uniforme sur $\mathcal{A}$
Résultat:
Générateur uniforme sur $\mathcal{B}$

$N = 0$
repeat
  $X =$**Génère-unif($\mathcal{A}$)**
  $N = N + 1$
until $X \in \mathcal{B}$
return X, N

Complexité

N Nombre d'itérations

$$\begin{aligned} \mathbb{P}(N = k) &= \mathbb{P}(X \in \mathcal{B}, N = k) \\ &= \left(1 - \frac{|\mathcal{B}|}{|\mathcal{A}|}\right)^{k-1} \frac{|\mathcal{B}|}{|\mathcal{A}|} \end{aligned}$$

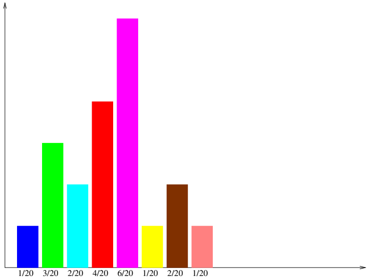
Loi géométrique de paramètre $p_a = \frac{|\mathcal{B}|}{|\mathcal{A}|}$.
Nombre moyen d'itérations

$$\begin{aligned} \mathbb{E}\, N &= \sum_{k=1}^{+\infty} k(1 - p_a)^{k-1} p_a \\ &= \frac{1}{(1 - (1 - p_a))^2} p_a = \frac{1}{p_a}. \end{aligned}$$

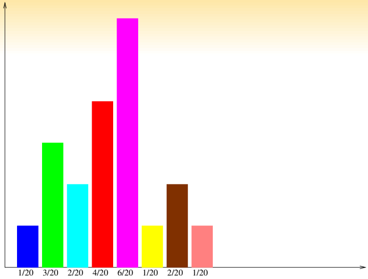
$$Var\ N = \frac{1 - p_a}{p_a^2}$$

Outline

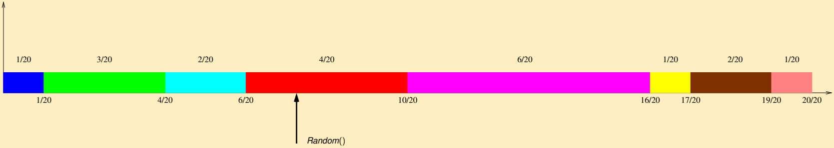
Loi sur un ensemble fini



Loi sur un ensemble fini

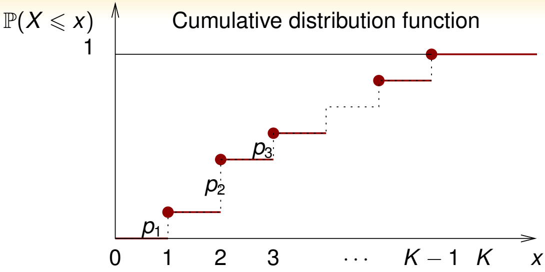


Histogramme : représentation "à plat"



$$\text{Coût (nombre moyen de comparaisons)} : \hat{C}(P) = \sum_{k=1}^{K} k.p_k = 4.35$$

Inverse de la fonction de répartition



Principe

Diviser $[0,1[$ en intervalles de longueur p_k
Trouver l'intervalle contenant *Random*
Retourner l'index de l'intervalle
Coût de calcul : $\mathcal{O}(\mathbb{E}X)$ itérations
Coût mémoire : $\mathcal{O}(K)$

Inverse de la fonction de répartition: algorithme

Algorithme

Inverse(P[])

Données: Un tableau de probabilités $P[] = \{p_1, \cdots, p_K\}$
Résultat: Un entier k généré avec la probabilité p_k

$u =$**Random()**
$k = 0$
$S = 0$
while $u > S$
  $k = k + 1$
  $S = S + P[k]$
return k

UNIVERSITE JOSEPH FOURIER SCIENCES. TECHNOLOGIE. SANTÉ

Searching optimization

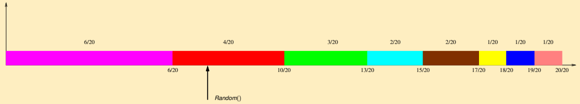
Optimization methods

- pre-compute the pdf in a table
- rank objects by decreasing probability

Searching optimization

Optimization methods

- pre-compute the pdf in a table
- rank objects by decreasing probability

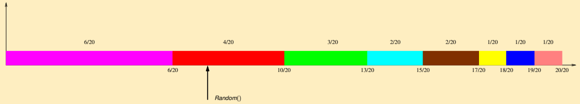


- use a dichotomy algorithm
- use a tree searching algorithm (optimality = Huffmann coding tree)

Searching optimization

Optimization methods

- pre-compute the pdf in a table
- rank objects by decreasing probability

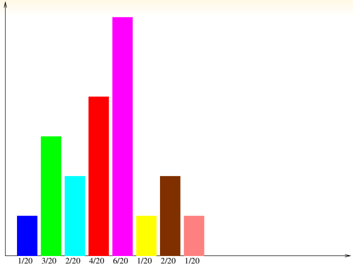


- use a dichotomy algorithm
- use a tree searching algorithm (optimality = Huffmann coding tree)

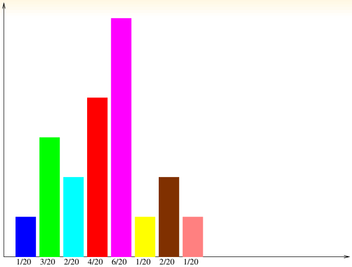
Comments

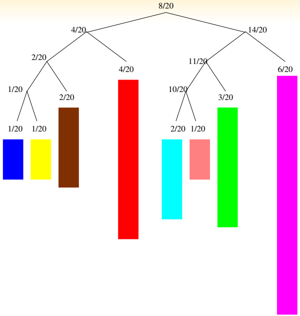
- Depends on the usage of the generator (repeated use or not)
- pre-computation usually $\mathcal{O}(K)$ could be huge
-

Optimalité

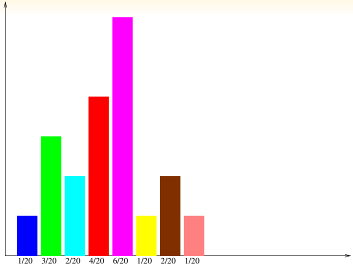


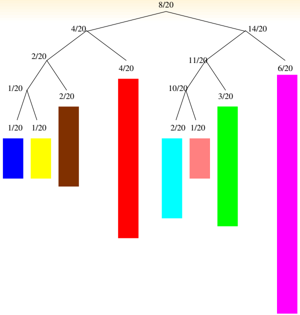
Optimalité


1/20
3/20
2/20
4/20
6/20
1/20
2/20
1/20


8/20
4/20
14/20
2/20
4/20
11/20
6/20
1/20
2/20
10/20
3/20
1/20
1/20
2/20
1/20

Optimalité



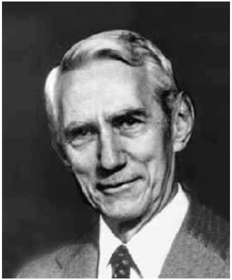


Nombre de comparaisons

Structure d'arbre binaire de recherche

$$\mathbb{E}N = \sum_{k=1}^{K} p_k.l_k = 3,75, \text{ Entropie} = \sum_{k=1}^{K} p_k(-\log_2 p_k) = 3.70$$

Claude Shannon (1916-2001)



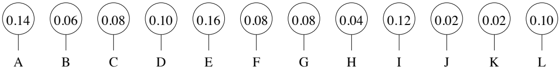
Claude Elwood Shannon (30 avril 1916 à Gaylord, Michigan - 24 février 2001), ingénieur électrique, est l'un des pères, si ce n'est le père fondateur, de la théorie de l'information. Son nom est attaché à un célèbre "schéma de Shannon" très utilisé en sciences humaines, qu'il a constamment désavoué.

Il étudia le génie électrique et les mathématiques à l'Université du Michigan en 1932. Il utilisa notamment l'algèbre booléenne pour sa maîtrise soutenue en 1938 au MIT. Il y expliqua comment construire des machines à relais en utilisant l'algèbre de Boole pour décrire l'état des relais (1 : fermé, 0 : ouvert).
Shannon travailla 20 ans au MIT, de 1958 à 1978. Parallèlement à ses activités académiques, il travailla aussi aux laboratoires Bell de 1941 à 1972.
Claude Shannon était connu non seulement pour ses travaux dans les télécommunications, mais aussi pour l'étendue et l'originalité de ses hobbies, comme la jonglerie, la pratique du monocycle et l'invention de machines farfelues : une souris mécanique sachant trouver son chemin dans un labyrinthe, un robot jongleur, un joueur d'échecs (roi tour contre roi)...
Souffrant de la maladie d'Alzheimer dans les dernières années de sa vie, Claude Shannon est mort à 84 ans le 24 février 2001.

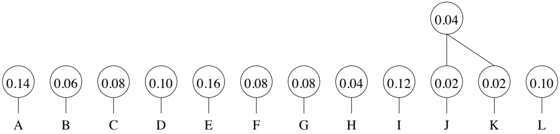
Wikipedia

Algorithme de Huffman (1951)

A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10



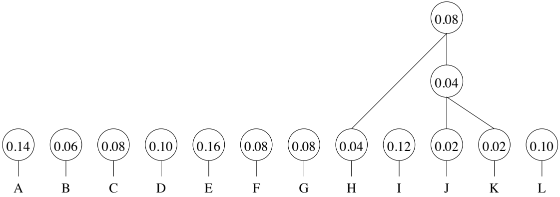
Algorithme de Huffman (1951)



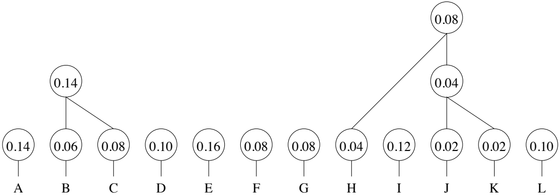
A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10

Algorithme de Huffman (1951)

A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10

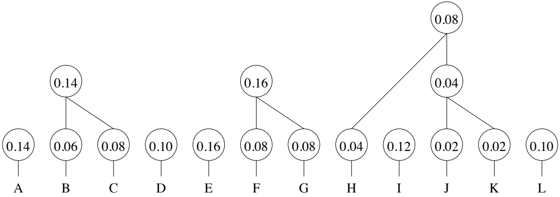


Algorithme de Huffman (1951)



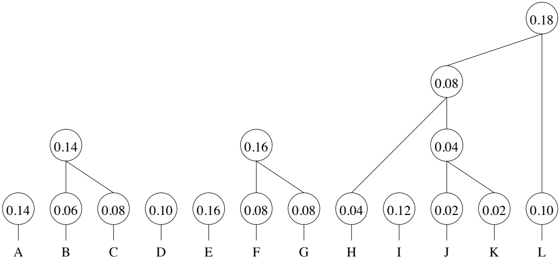
A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10

Algorithme de Huffman (1951)



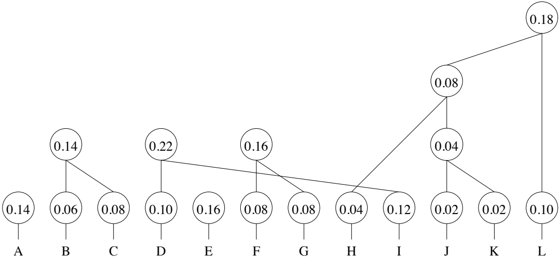
A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10

Algorithme de Huffman (1951)



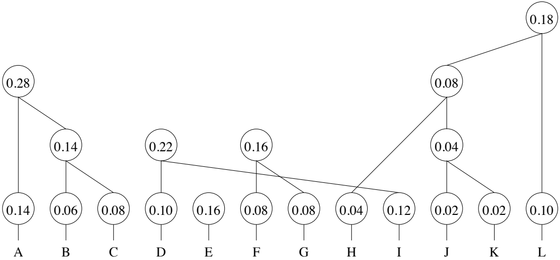
A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10

Algorithme de Huffman (1951)



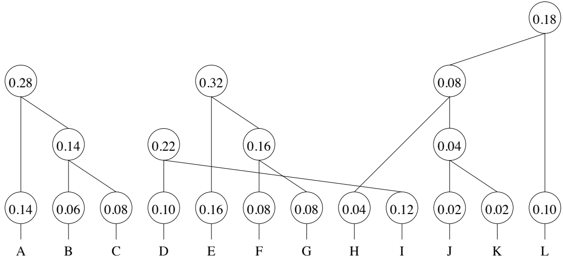
A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10

Algorithme de Huffman (1951)



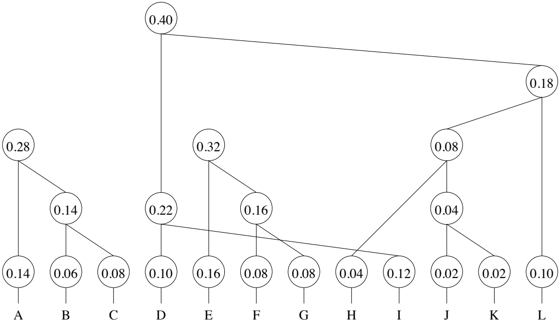
A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10

Algorithme de Huffman (1951)



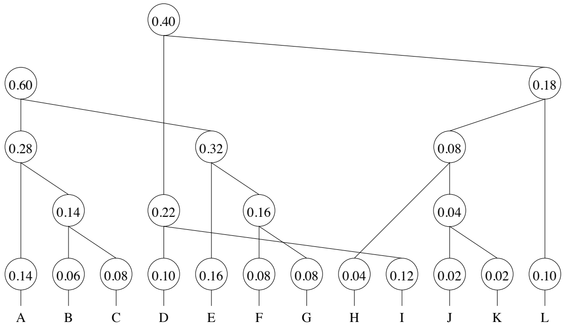
A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10

Algorithme de Huffman (1951)



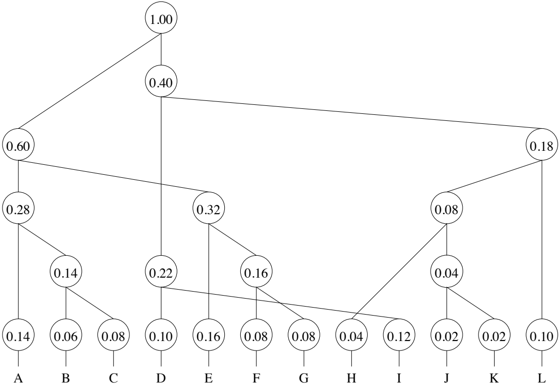
A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10

Algorithme de Huffman (1951)



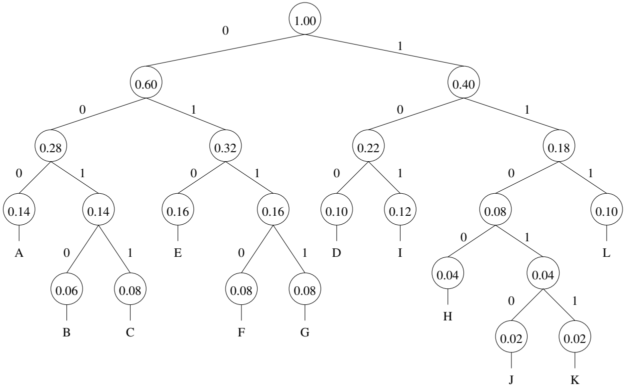
A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10

Algorithme de Huffman (1951)



A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10

Algorithme de Huffman (1951)



A	0.14	000
B	0.06	0010
C	0.08	0011
D	0.10	100
E	0.16	010
F	0.08	0110
G	0.08	0111
H	0.04	1100
I	0.12	101
J	0.02	11010
K	0.02	11011
L	0.10	111

Codage optimal : L-moy = 3.42, Entropie = 3.38
Profondeur = $-\log_2$(probabilité)
Généralisation Lempel-Ziv,...

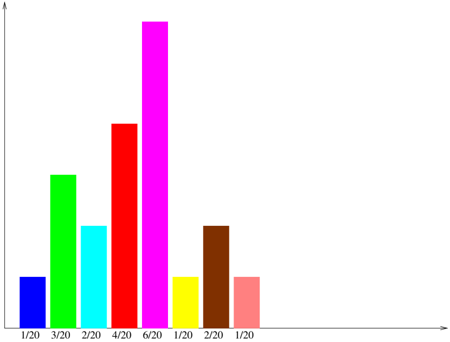
Algorithme de Huffman (1951): Implantation

Arbre-Huffman(P[])

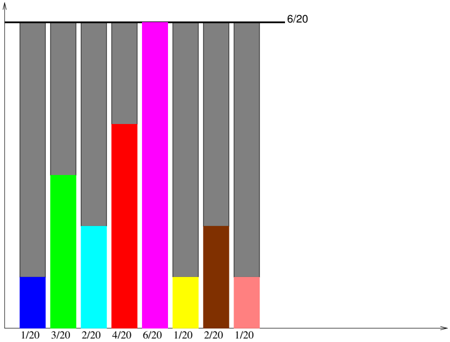
Données: Un tableau de probabilité P=$\{p_1, \cdots, p_K\}$
Résultat: Un arbre binaire de Huffman transformé en arbre binaire de recherche

```
F: file à priorité
for k = 1 to K
    z=nouveau_noeud()
    z.gauche=Nil z.droit=Nil
    z.poids=P[k] Insérer(F,z)
while Taille(F)≠1
    z=nouveau_noeud()
    z.gauche=Extraire(F) z.droit=Extraire(F)
    z.poids=z.gauche.poids+z.droit.poids Insérer(F,z)
z=Extraire(F)
Mettre_à_jour_étiquettes(z)                // parcours infixé
```

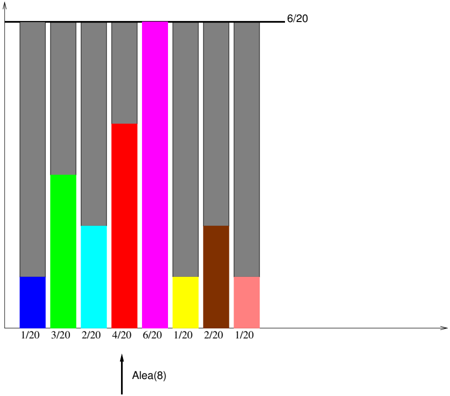
Méthode basée sur le rejet



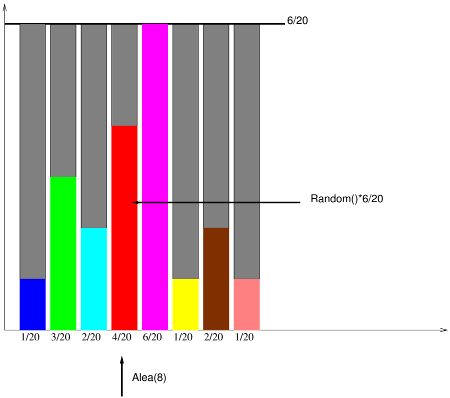
Méthode basée sur le rejet



Méthode basée sur le rejet



Méthode basée sur le rejet



Méthode basée sur le rejet (suite)

Génère(P[])

Données: Un tableau de probabilités $P[] = \{p_1, \cdots, p_K\}$
Résultat: Un entier k généré avec la probabilité p_k

$N = 0$
repeat
  $k =$ Partie entière$(Random() * K + 1)$
  $u = Random() * p_{max}$
  $N = N + 1$
until $u \leqslant P[k]$
return k, N

Preuve

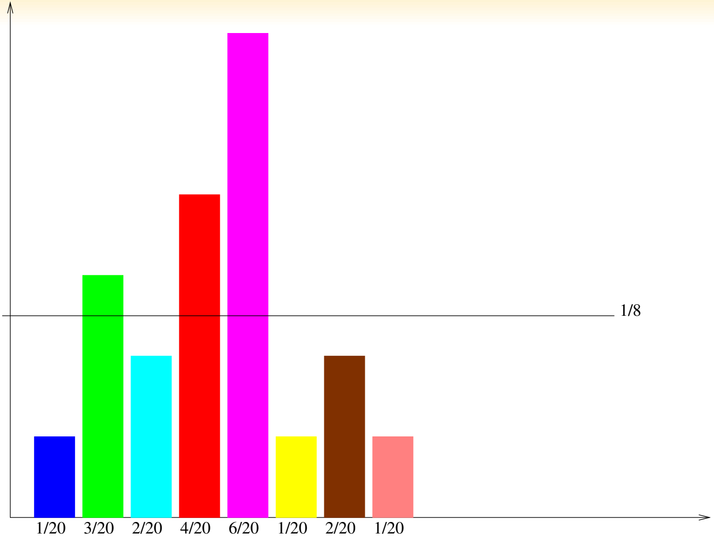
Même preuve que pour la loi uniforme

Complexité

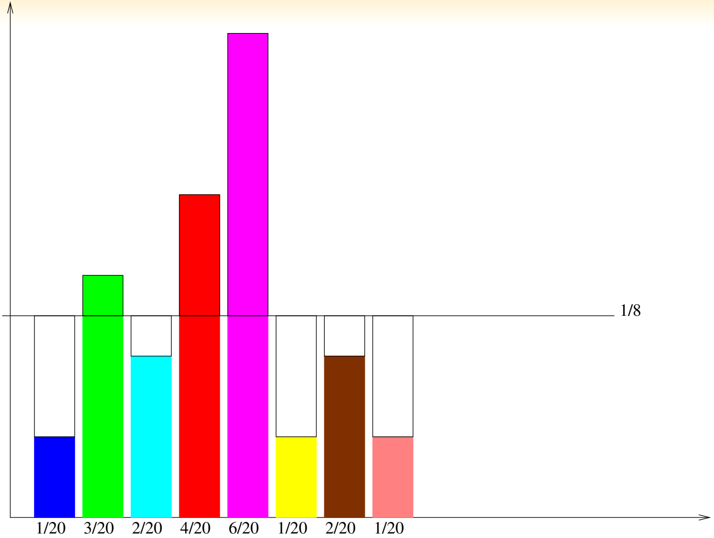
Coût moyen en nombre d'itérations :

$$p_a = \frac{1}{K.p_{max}} \text{ et } \mathbb{E}N = Kp_{max}$$

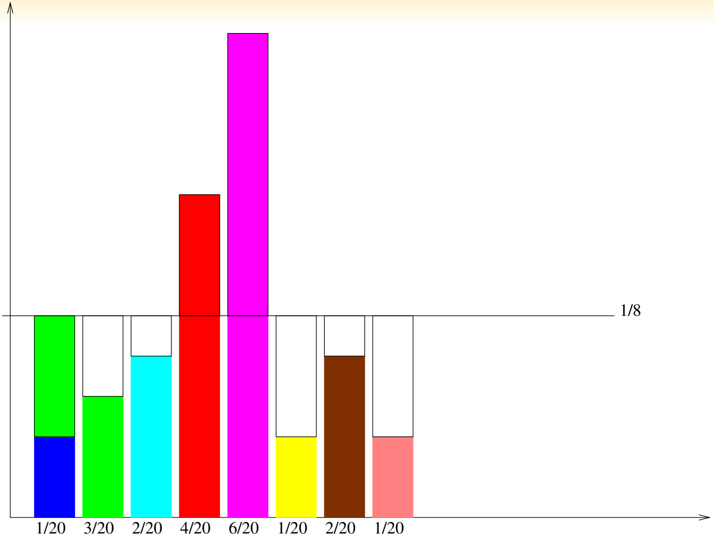
Méthode d'aliasing



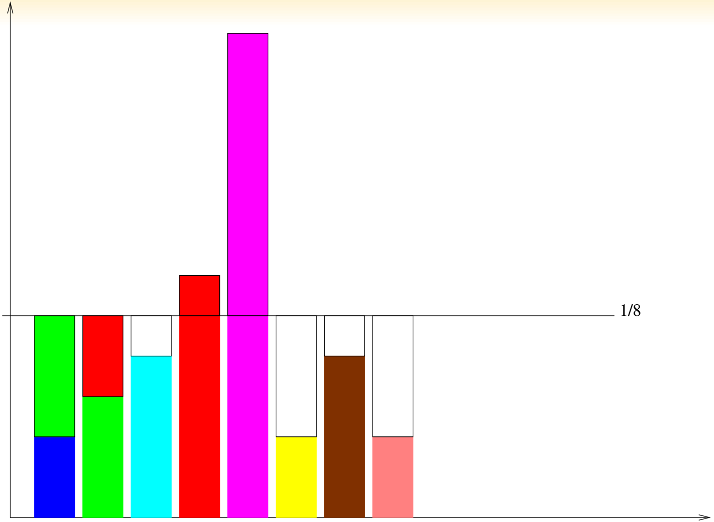
Méthode d'aliasing



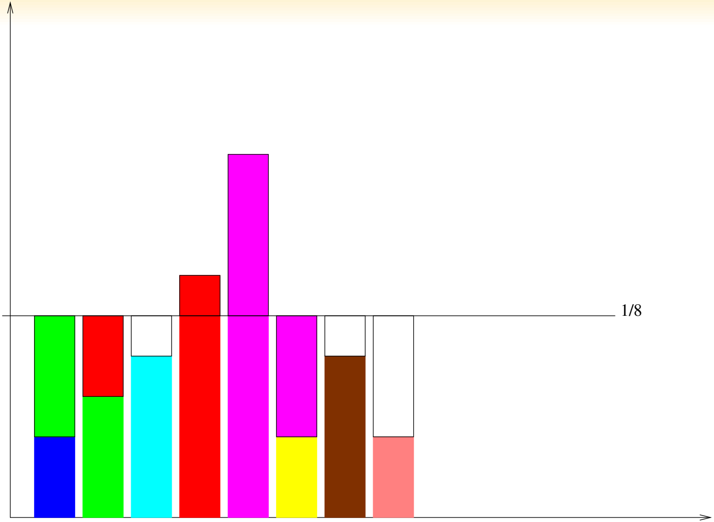
Méthode d'aliasing



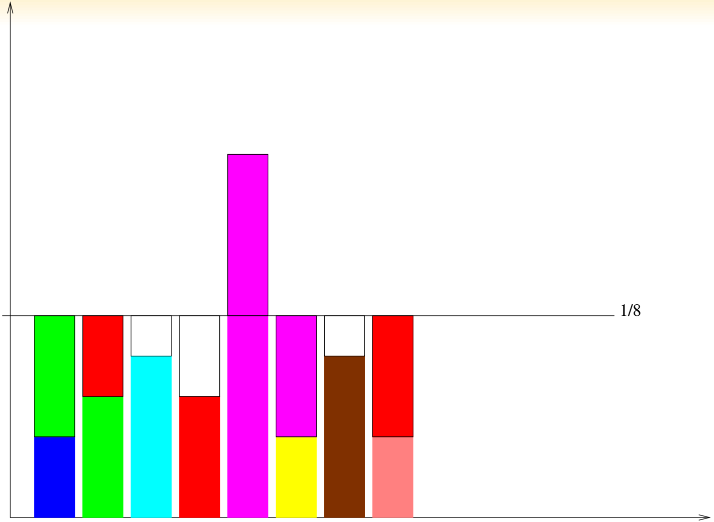
Méthode d'aliasing



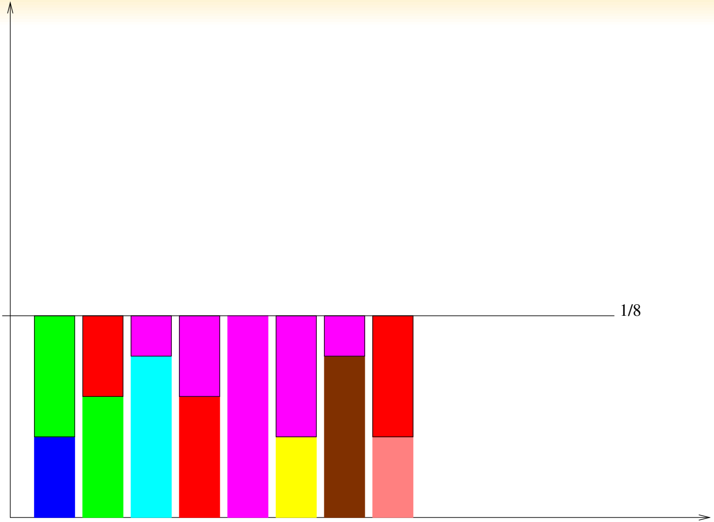
Méthode d'aliasing



Méthode d'aliasing



Méthode d’aliasing



Méthode d'aliasing : construction des tables

Table_Alias(P[])

Données: Un tableau de probabilité $P = [p_1, \cdots, p_K]$
Résultat: Un tableau de seuils $S = [s_1, \cdots, s_K]$
et d'alias $A = [a_1, \cdots, a_K]$

$L = \emptyset$ $U = \emptyset$
for $k = 1$ **to** K
    switch $P[k]$ **do**
        case $< \frac{1}{K}$ $L = L \cup \{k\}$
        case $> \frac{1}{K}$ $U = U \cup \{k\}$

while $L \neq \emptyset$
    $i = Extract(L)$ $k = Extract(U)$
    $S[i] = P[i]$ $A[i] = k$
    $P[k] = P[k] - (\frac{1}{K} - P[i])$
    switch $P[k]$ **do**
        case $< \frac{1}{K}$ $L = L \cup \{k\}$
        case $> \frac{1}{K}$ $U = U \cup \{k\}$

Méthode d'aliasing : génération

Génère(S[],A[])

Données: Un tableau de seuils $S = [s_1, \cdots, s_K]$
et d'alias $A = [a_1, \cdots, a_K]$
Résultat: Un indice k généré selon la probabilité $P = [p_1, \cdots, p_K]$

k = **Alea**(K) `// générateur uniforme d'entiers de` 1 `à` K
if Random() $\frac{1}{K} < S[k]$
  return k
else
  return $A[k]$

Complexité

Temps de calcul :
- $\mathcal{O}(K)$ pour le pré-calcul des tables d'alias
- $\mathcal{O}(1)$ pour la génération

Mémoire :
- seuils $\mathcal{O}(K)$ (même coût que le vecteur P)
- alias $\mathcal{O}(K)$ (tableau d'index)

Outline

Synthèse

Méthodes de base

- Inverse de la fonction de répartition (avec optimisation)
  forme "close" de l'inverse de la fonction de répartition
- Méthode à base de rejet
  Loi proche de l'uniforme, p_k faciles à calculer
- Méthode d'aliasing
  pré-calcul (coût amorti), surcoût mémoire
- Méthodes de composition
  dépend de la structure des valeurs de probabilité
- et d'autres encore...

Remarques

- temps d'exécution dépend de l'architecture de la machine
- algorithme dépend souvent du problème sous-jacent à simuler (exemple Chevalier de Méré)
- rechercher la loi uniforme...