Discrete Random Simulation

Flipping a coin or more

Jean-Marc.Vincent@univ-grenoble-alpes.fr

University de Grenoble-Alpes, UFR IM^{2}AG
MOSIG 1 Mathematics for Computer Science

November 2018

Story of Dice

Coins, dice wheels, ...

Physical mechanism :
Sequence of observations : $x_1, x_2, x_3, \cdots, x_n, \cdots$ in $\{1, 2, \cdots, K\}$
Probabilistic model
The sequence of observations is modeled by a sequence of

- random variables,
- independent,
- identically distributed,
- with a uniform distribution on the set $\{1, 2, \cdots, K\}$ denoted by $\{X_n\}_{n\in\mathbb{N}}$

Notations and properties

For all n and for all sequence in $\{x_1, \cdots, x_n\}$ in $\{1, 2, \cdots, K\}^n$

$$\begin{aligned}
\mathbb{P}(X_1 = x_1, \cdots, X_n = x_n) &= \mathbb{P}(X_1 = x_1). \cdots .\mathbb{P}(X_n = x_n) \text{ independence;} \\
&= \mathbb{P}(X = x_1). \cdots .\mathbb{P}(X = x_n) \text{ same distribution;} \\
&= \frac{1}{K} \cdots \frac{1}{K} = \frac{1}{K^n} \text{ uniform law.}
\end{aligned}$$

DICE STORY (CONT.)

Coin $\mapsto$ Dice-8

From throws of coins simulate a 8 faces dice :

DICE STORY (CONT.)

Coin $\mapsto$ Dice-8

From throws of coins simulate a 8 faces dice :

Dice-8()

Data: Function "Coin()" uniform generator in $\{0, 1\}$
Result: A sequence i.i.d. variables uniform on $\{1, \cdots, 8\}$

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

TALES OF DICE : PROOF OF THE ALGORITHMS

Specification :
a sequence of calls of **Dice-8()** function is modeled by a sequence of random variables independent and identically distributed (i.i.d.) with uniform probability law on $\{1, \cdots, 8\}$.

TALES OF DICE : PROOF OF THE ALGORITHMS

Specification :
a sequence of calls of **Dice-8()** function is modeled by a sequence of random variables independent and identically distributed (i.i.d.) with uniform probability law on $\{1, \cdots, 8\}$.

Hypothesis :
$C_0, C_1, \cdots, C_n, \cdots$ sequence of calls to **Coin()** i.i.d. sequence uniform on $\{0, 1\}$

TALES OF DICE : PROOF OF THE ALGORITHMS

Specification :
a sequence of calls of **Dice-8()** function is modeled by a sequence of random variables independent and identically distributed (i.i.d.) with uniform probability law on $\{1, \cdots, 8\}$.
Hypothesis :
$C_0, C_1, \cdots, C_n, \cdots$ sequence of calls to **Coin()** i.i.d. sequence uniform on $\{0, 1\}$
Preuve :
Denote by $S_0, S_1, \cdots, S_n, \cdots$ the sequence of random variables modeling the results obtained by the successive calls to **Dice-8()** .
Let $n \in \mathbb{N}$ and $(x_0, x_1, \cdots, x_n) \in \{1, \cdots, 8\}^{n+1}$. We should show that

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

Tales of Dice : Proof of the algorithms (2)

We have

$$\begin{aligned}
\mathbb{P}(S_0 = x_0, \cdots, S_n = x_n) & \\
= \quad & \mathbb{P}(S_0 = x_0) \cdots \mathbb{P}(S_n = x_n) \\
& \text{car } S_k \text{ depends only on } C_{3k}, C_{3k+1}, C_{3k+2} \text{ and } C_i \text{ are independent;} \\
& \text{les } S_0, \cdots, S_n, \cdots \text{ are indépendent;} \\
= \quad & \mathbb{P}(S_0 = x_0) \cdots \mathbb{P}(S_0 = x_n) \text{ because } (C_{3k}, C_{3k+1}, C_{3k+2}) \text{ have the same law}
\end{aligned}$$

TALES OF DICE : PROOF OF THE ALGORITHMS (2)

We have

$$\begin{aligned}
&\mathbb{P}(S_0 = x_0, \cdots, S_n = x_n) \\
&\quad = \mathbb{P}(S_0 = x_0) \cdots \mathbb{P}(S_n = x_n) \\
&\qquad \text{car } S_k \text{ depends only on } C_{3k}, C_{3k+1}, C_{3k+2} \text{ and } C_i \text{ are independent;} \\
&\qquad \text{les } S_0, \cdots, S_n, \cdots \text{ are indépendent;} \\
&\quad = \mathbb{P}(S_0 = x_0) \cdots \mathbb{P}(S_0 = x_n) \text{ because } (C_{3k}, C_{3k+1}, C_{3k+2}) \text{ have the same law}
\end{aligned}$$

But for i dans $\{1, \cdots, 8\}$, $i-1$ has a unique binary decomposition $i - 1 =_2 a_2 a_1 a_0$.

$$\begin{aligned}
\mathbb{P}(S_0 = i) &= \mathbb{P}(C_0 = a_0, C_1 = a_1, C_2 = a_2) \\
&= \mathbb{P}(C_0 = a_0)\mathbb{P}(C_1 = a_1)\mathbb{P}(C_2 = a_2) \text{ calls to Coin() are independent;} \\
&= \frac{1}{2}\frac{1}{2}\frac{1}{2} = \frac{1}{8} \text{ have the same law on } \{0, 1\}.
\end{aligned}$$

then

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

TALES OF DICE (3)

Coin $\mapsto$ Dice-2^k

From one coin design a random generator of a 2^k-sided dice.

TALES OF DICE (3)

Coin $\mapsto$ Dice-2^k

From one coin design a random generator of a 2^k-sided dice.

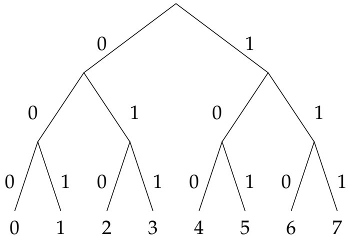
Dice(k)

Data: A function "Coin()" random generator on $\{0, 1\}$
Result: A sequence of iid numbers uniformly distributed on $\{1, \cdots, 2^k\}$

S=0
for $i = 1$ **to** k
  S=Coin() +2.S `// cf Hörner's Scheme`
$S = S + 1$
return S

Preuve: Same proof as for **Dice-8**, based on the unicity of the binary decomposition of an integer in $\{0, \cdots, 2^k - 1\}$ by a k bits vector.

BINARY REPRESENTATION :



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

TALES OF DICE (4)

Coin $\mapsto$ Dice-6

From a coin design a 6-sided dice.

TALES OF DICE (4)

Coin $\mapsto$ Dice-6

From a coin design a 6-sided dice.

Dice-6()

Data: A function **Dice-8()** random generator on $\{1, \cdots, 8\}$
Result: A sequence of i.i.d. random numbers uniformly distributed on $\{1, \cdots, 6\}$

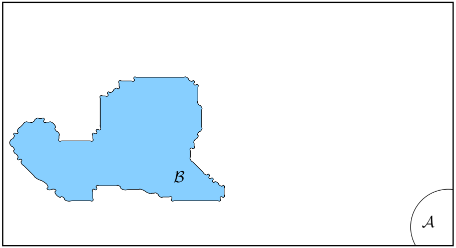
repeat
| $X =$**Dice-8()**
until $X \leqslant 6$
return X

Proof: later

GENERATION METHODS BASED ON REJECTION

Principe

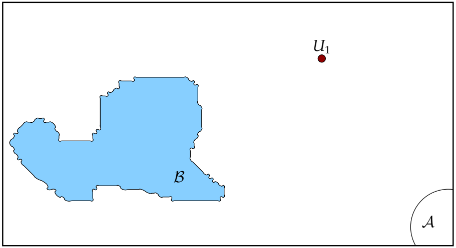
Generate uniformly on $\mathcal{A}$ accept if the point is in $\mathcal{B}$.



GENERATION METHODS BASED ON REJECTION

Principe

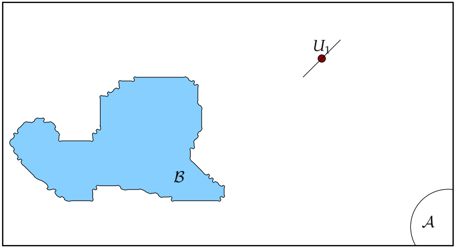
Generate uniformly on $\mathcal{A}$ accept if the point is in $\mathcal{B}$.



Generation Methods based on Rejection

Principe

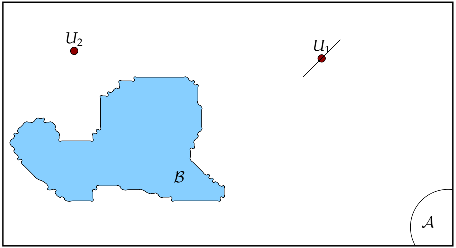
Generate uniformly on $\mathcal{A}$ accept if the point is in $\mathcal{B}$.



GENERATION METHODS BASED ON REJECTION

Principe

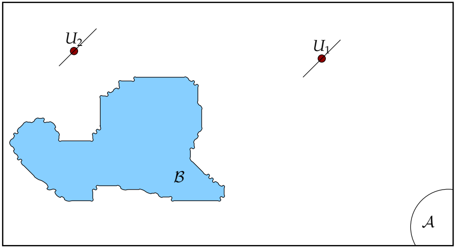
Generate uniformly on $\mathcal{A}$ accept if the point is in $\mathcal{B}$.



Generation Methods based on Rejection

Principe

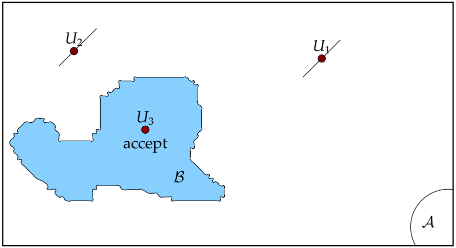
Generate uniformly on $\mathcal{A}$ accept if the point is in $\mathcal{B}$.



GENERATION METHODS BASED ON REJECTION

Principe

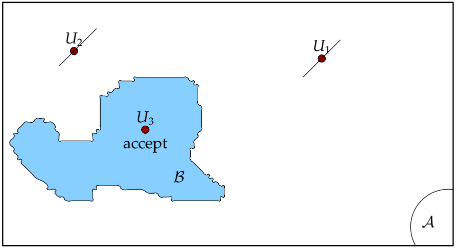
Generate uniformly on $\mathcal{A}$ accept if the point is in $\mathcal{B}$.



Generation Methods based on Rejection

Principe

Generate uniformly on $\mathcal{A}$ accept if the point is in $\mathcal{B}$.



Algorithm

Generation-unif($\mathcal{B}$)

Data:
Uniform generator on $\mathcal{A}$
Result:
Uniform generator on $\mathcal{B}$

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

GENERATION METHODS BASED ON REJECTION: PROOF

Génère-unif($\mathcal{B}$)

Data:
Uniform generator on $\mathcal{A}$
Result:
Uniform generator on $\mathcal{B}$

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

Proof

Calls to **Generation-unif($\mathcal{A}$)**: $X_1, X_2, \cdots, X_n, \cdots$

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

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

Consequently the law is **uniform** on $\mathcal{B}$

GENERATION METHODS BASED ON REJECTION

Génère-unif($\mathcal{B}$)

Data:
Uniform generator on $\mathcal{A}$
Result:
Uniform generator on $\mathcal{B}$

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

Complexity

N Number of iterations

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

Geometric probability distribution with parameter $p_a = \frac{|\mathcal{B}|}{|\mathcal{A}|}$.
Expected number of iterations

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

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

GENERATING RANDOM OBJECTS

Denote by X the generated object $\in \{1, \cdots, n\}$
Distribution (proportion of observations, input of the load injector)

$$p_k = \mathbb{P}(X = k).$$

GENERATING RANDOM OBJECTS

Denote by X the generated object $\in \{1, \cdots, n\}$
Distribution (proportion of observations, input of the load injector)

$$p_k = \mathbb{P}(X = k).$$

Remarks :

$$0 \leqslant p_i \leqslant 1; \quad \sum_k p_k = 1.$$

GENERATING RANDOM OBJECTS

Denote by X the generated object $\in \{1, \cdots, n\}$
Distribution (proportion of observations, input of the load injector)

$$p_k = \mathbb{P}(X = k).$$

Remarks :

$$0 \leqslant p_i \leqslant 1; \quad \sum_k p_k = 1.$$

For integer valued random variables $X \in \mathbb{N}$:

$$\mathbb{E}X = \sum_k k.\mathbb{P}(X = k) = \sum_k kp_k.\text{Expectation}$$

Variance and standard deviation

$$\mathbb{V}arX = \sum_k (k - \mathbb{E}X)^2 \mathbb{P}(X = k) = \mathbb{E}(X - \mathbb{E}X)^2 = \mathbb{E}X^2 - (\mathbb{E}X)^2.$$

$$\sigma(X) = \sqrt{\mathbb{V}arX}.$$

THE RANDOM FUNCTION

Random bit generator (see previous lecture)

double drand48(void) (48 bits encoded in 8 bytes) *(manpage)*

The rand48() family of functions generates pseudo-random numbers using a linear congruential algorithm working on integers 48 bits in size. The particular formula employed is r(n+1) = (a * r(n) + c) mod m where the default values are for the multiplicand a = 0xfdeece66d = 25214903917 and the addend c = 0xb = 11. The modulo is always fixed at m = 2 ** 48. r(0) is called the seed of the random number generator.

The sequence of returned values from a sequence of calls to the random function is modeled by a sequence of real independent random variables uniformly distributed on the real interval $[0, 1)$

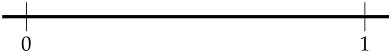
Probabilistic Model

$\{U_n\}_{n\in\mathbb{N}}$ sequence of i.i.d real random variables

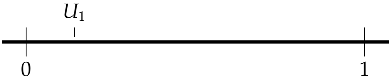
For all $n \in \mathbb{N}$, for all the intervals $[a_i, b_i)$ with $0 \leqslant i \leqslant n$ and $0 \leqslant a_i < b_i \leqslant 1$,

$$\mathbb{P}\left(U_0 \in [a_0, b_0), \cdots, U_n \in [a_n, b_n)\right) = (b_0 - a_0) \times \cdots \times (b_n - a_n).$$

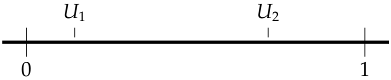
The random function



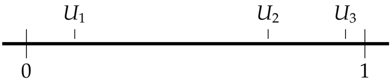
The random function



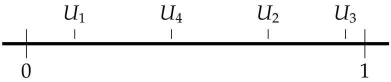
The random function



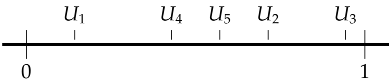
The random function



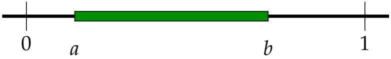
THE RANDOM FUNCTION



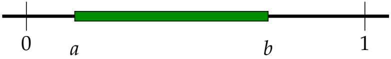
The random function



The random function

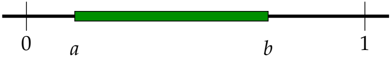


The random function



$$\mathbb{P}(U \in [a, b[) = (b - a)$$

THE RANDOM FUNCTION



$\mathbb{P}(U \in [a, b[) = (b - a)$

Problem

All the difficulty is to find a function (an algorithm) that maps the $[0, 1[$ in a set with a right probability.

UNIFORM DISCRETE RANDOM VARIABLES

Example : flip a coin

Coin ()
  u=`Random` () **if** $u < \frac{1}{2}$
    Return 0 `// or returns Head`
  else
    `Return` 1 `// or returns Tail`

Bernoulli scheme

Roll a n-sided dice

Dice (n)
  Data: n : Number of possible outcomes
  Result: a single outcome in $\{1, \cdots, n\}$
  u=`Random` ()
  `Return` $\lceil n * u \rceil$
  `// smallest integer greater than` $u \times n$

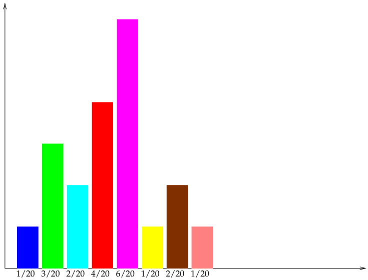
The problem

Given a discrete distribution

$$p = (p_1, p_2, \cdots, p_n), \quad 0 \leqslant p_i \leqslant 1 \quad \sum_{i=1}^{n} p_i = 1;$$

Design an algorithm that generates pseudo random numbers according probability p.
Prove such an algorithm and evaluate its (average) **complexity**

PROBABILITIES ON A FINITE SET



TABULATION METHOD

Pre-computation

$$p_k = \frac{m_k}{m} \text{ where } \sum_k m_k = m.$$

Create a table T with size m.
Fill T such that m_k cells contains k.
Computation cost : m steps
Memory cost : m

TABULATION METHOD

Pre-computation

$$p_k = \frac{m_k}{m} \text{ where } \sum_k m_k = m.$$

Create a table T with size m.
Fill T such that m_k cells contains k.
Computation cost : m steps
Memory cost : m

Table construction

```
Build_Table (p)
    Data: p a rational distribution p_i = m_i / m
    Result: Tabulation of distribution p
    l=1
    for i = 1,  i ⩽ n,  i++
        for j = 1,  j ⩽ m_i  j++
            T[l]=i
            l++
```

TABULATION METHOD

Pre-computation

$$p_k = \frac{m_k}{m} \text{ where } \sum_k m_k = m.$$

Create a table T with size m.
Fill T such that m_k cells contains k.
Computation cost : m steps
Memory cost : m

Table construction

```
Build_Table (p)
   Data: p a rational distribution p_i = m_i/m
   Result: Tabulation of distribution p
   l=1
   for i = 1, i ⩽ n, i++
      for j = 1, j ⩽ m_i j++
         T[l]=i
         l++
```

Generation

Generate uniformly on the set $\{1, \cdots, m\}$
Returns the value in the table
Computation cost : $\mathcal{O}(1)$ step
Memory cost : $\mathcal{O}(m)$

TABULATION METHOD

Pre-computation

$$p_k = \frac{m_k}{m} \text{ where } \sum_k m_k = m.$$

Create a table T with size m.
Fill T such that m_k cells contains k.
Computation cost : m steps
Memory cost : m

Table construction

Build_Table (p)
Data: p a rational distribution $p_i = \frac{m_i}{m}$
Result: Tabulation of distribution p

```
l=1
for i = 1, i ⩽ n, i++
    for j = 1, j ⩽ m_i j++
        T[l]=i
        l++
```

Generation

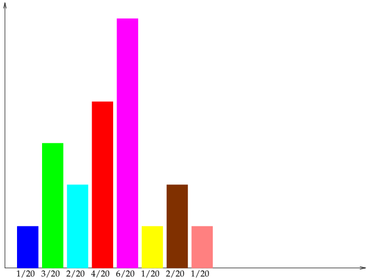
Generate uniformly on the set $\{1, \cdots, m\}$
Returns the value in the table
Computation cost : $\mathcal{O}(1)$ step
Memory cost : $\mathcal{O}(m)$

Generation algorithm

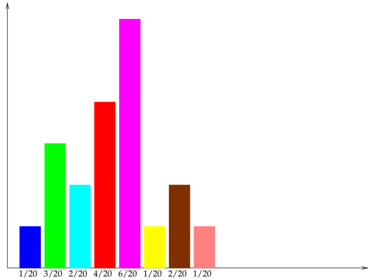
Generation (T)
Data: T Tabulation of distribution p
Result: A random number following distribution p

```
u=Random ()
l = ⌈m * u⌉
Return T[l]
```

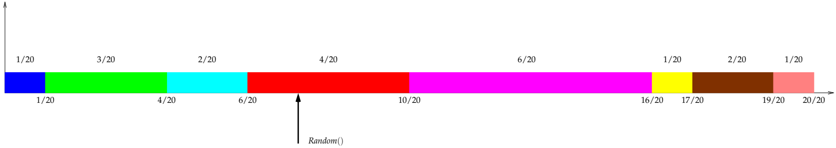
PROBABILITIES ON A FINITE SET



PROBABILITIES ON A FINITE SET

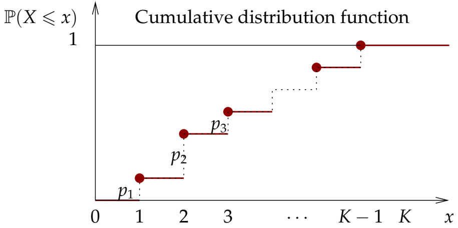


Histogram : Flat representation



Cost(average number of comparisons) : $\hat{C}(P) = \sum_{k=1}^{K} k.p_k = 4.35$

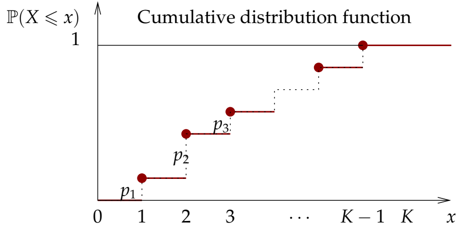
INVERSE OF PROBABILITY DISTRIBUTION FUNCTION



Generation

Divide $[0,1[$ in intervals with length p_k
Find the interval in which *Random* falls
Returns the index of the interval
Computation cost : $\mathcal{O}(\mathbb{E}X)$ steps
Memory cost : $\mathcal{O}(1)$

INVERSE OF PROBABILITY DISTRIBUTION FUNCTION



Generation

Divide $[0, 1[$ in intervals with length p_k
Find the interval in which *Random* falls
Returns the index of the interval
Computation cost : $\mathcal{O}(\mathbb{E}X)$ steps
Memory cost : $\mathcal{O}(1)$

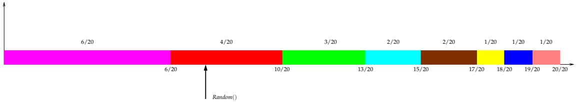
Inverse function algorithm

Generation (p)
Data: A distribution p
Result: A random number following distribution p
$u =$`Random` ()
$S = 0$
$k = 0$
while $u > s$
  $k = k + 1$
  $s = s + p_k$
`Return` k

SEARCHING OPTIMIZATION

Optimization methods

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

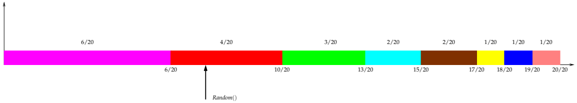


- use a dichotomy algorithm
- use a tree searching algorithm (optimality = Huffmann coding tree)

SEARCHING OPTIMIZATION

Optimization methods

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

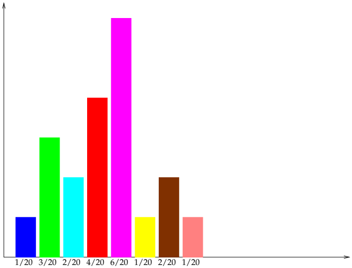


- use a dichotomy algorithm
- use a tree searching algorithm (optimality = Huffmann coding tree)

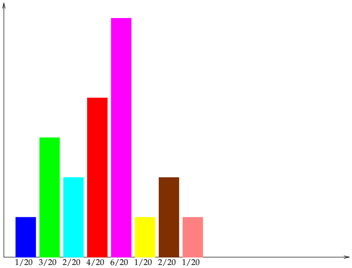
Comments

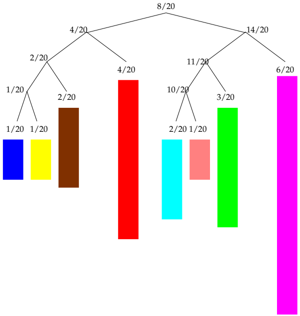
- Depends on the usage of the generator (repeated use or not)
- pre-computation usually $\mathcal{O}(K)$ could be huge
-

Optimality

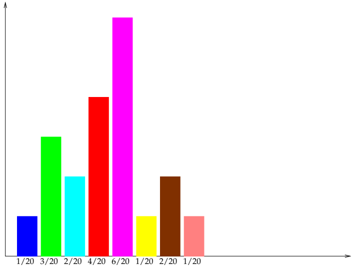


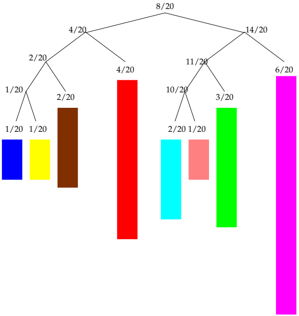
OPTIMALITY


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

Optimality



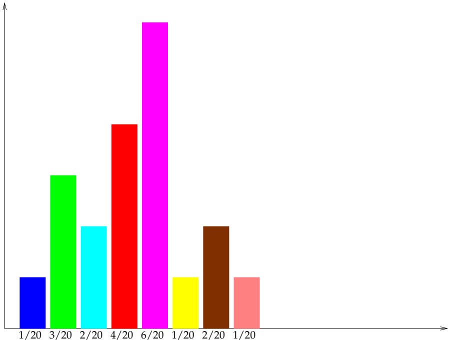


Number of comparisons

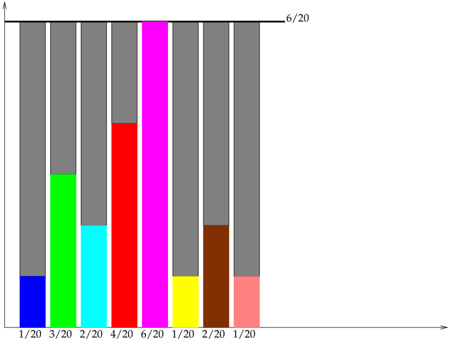
Binary Search Tree Structure

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

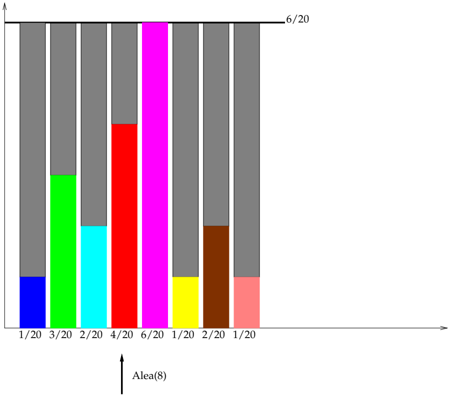
Rejection Based Methods



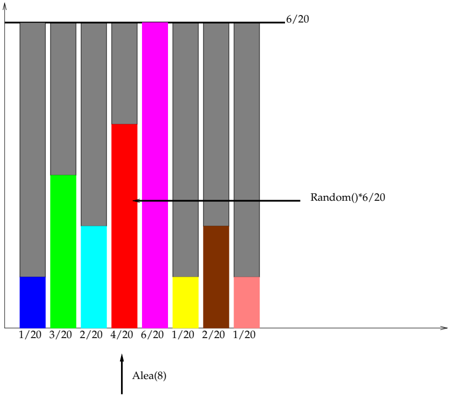
Rejection Based Methods



Rejection Based Methods



Rejection Based Methods



REJECTION BASED METHODS

Generation_Reject(p)

Data: A distribution p
Result: A random number following distribution p

$N = 0$
repeat
  $u =$`Random` ()
  $k = \lceil n * u \rceil$
  $v = Random() * p_{max}$
  $N++$
until $v \leqslant p_k$
`Return` k, N

Proof

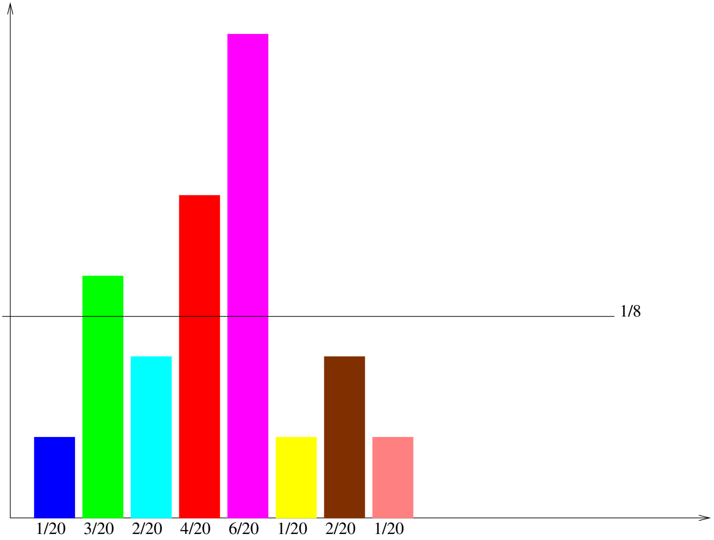
Same proof as for the uniform case

Complexity

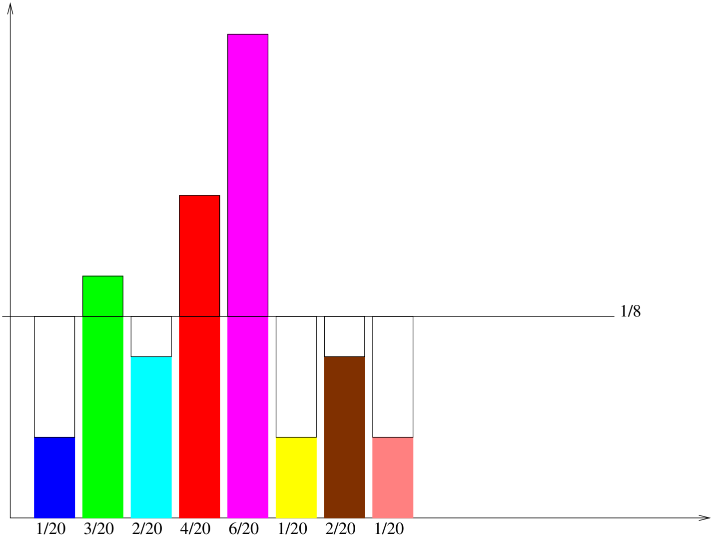
Average number of iterations :

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

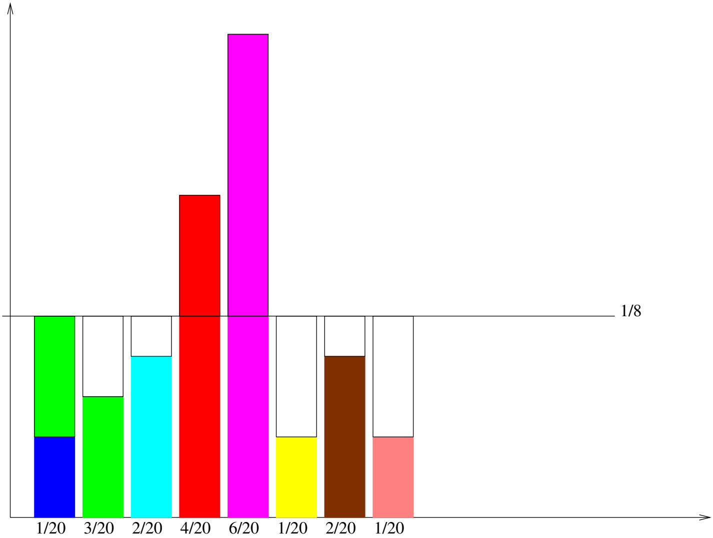
Aliasing Method



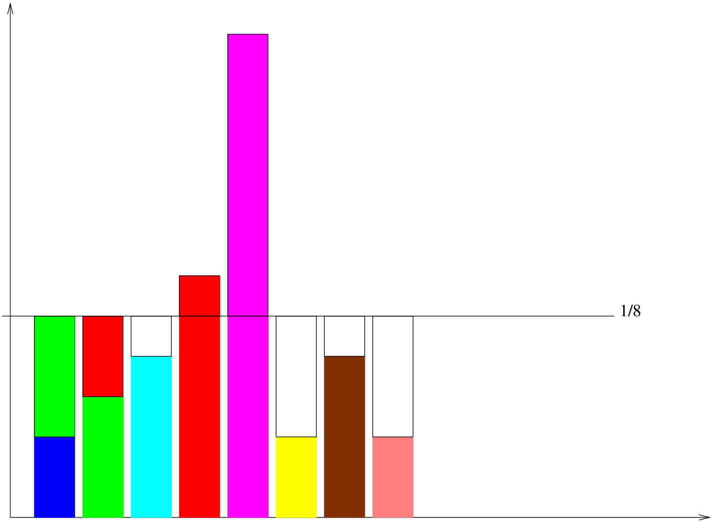
Aliasing Method



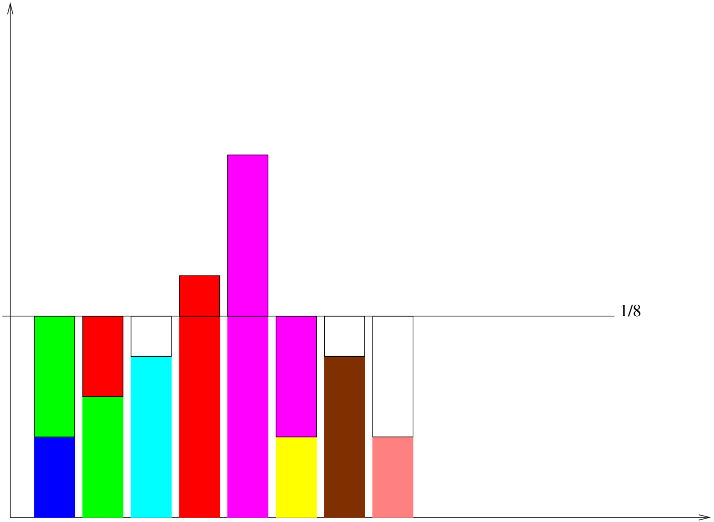
ALIASING METHOD



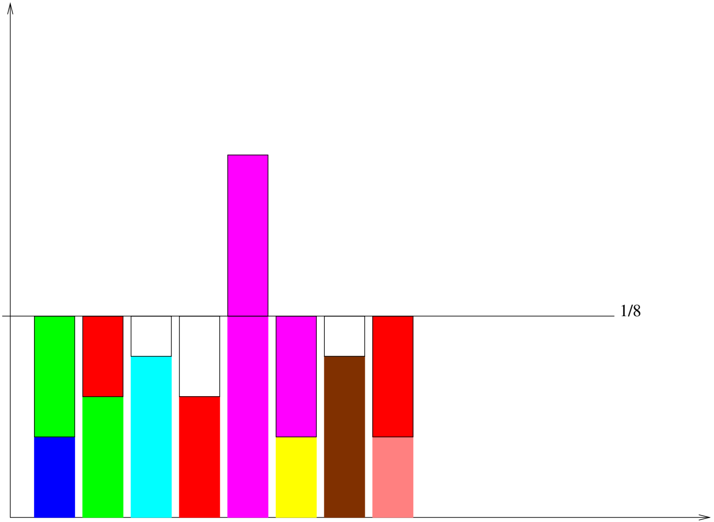
Aliasing Method



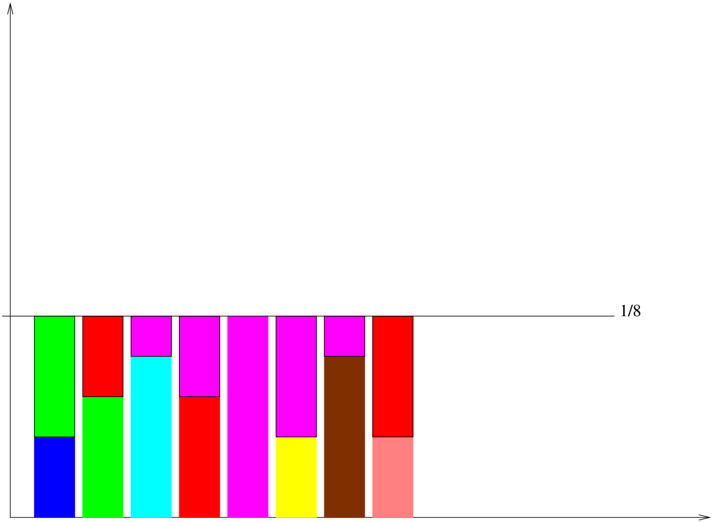
Aliasing Method



Aliasing Method



Aliasing Method



ALIASING METHOD : ALIAS TABLE

Table_Alias(p)

Data: A distribution p
Result: A vector of thresholds $s_1, \cdots, s_n]$
and a vector of aliases $a_1, \cdots, a_n$

$L = \emptyset$ $U = \emptyset$
for $k = 1$ **to** n
  switch p_k **do**
    case $p_k < \frac{1}{n}$ $L = L \cup \{k\}$
    case $p_k > \frac{1}{n}$ $U = U \cup \{k\}$
while $L \neq \emptyset$
  $i = Extract(L)$ $k = Extract(U)$
  $s_i = p_i$ $a_i = k$
  $p_k = p_k - (\frac{1}{n} - p_i)$
  switch p_k **do**
    case $< \frac{1}{n}$ $L = L \cup \{k\}$
    case $> \frac{1}{n}$ $U = U \cup \{k\}$

UNIVERSITÉ Grenoble Alpes

Aliasing Method : generation

Generation_Alias(s, a)

Data: A vector of thresholds $s_1, \cdots, s_n]$
and a vector of aliases $a_1, \cdots, a_n$ according adistribution p
Result: A random number following distribution p

u =`Random` ()
k = $\lceil n * u \rceil$
if Random() $\frac{1}{n} < s_k$
  `Return` k
else
  `Return` a_k

Complexity

Computation time :
- $\mathcal{O}(n)$ computation of thresholds and aliases
- $\mathcal{O}(1)$ generation
Memory :
- thresholds $\mathcal{O}(n)$ (same cost as p)
- alias $\mathcal{O}(n)$ (aliases)