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 2024

1. **Uniform : Uniform Random Variable**
2. Discrete : Discrete Random Variable
3. Uniform : Combinatorial Objects

Story of Dice

Coins, dice wheels, ... : a 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 modeled by a sequence of 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

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

TALES OF DICE : PROOF OF THE ALGORITHMS (2)

We have

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

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

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.

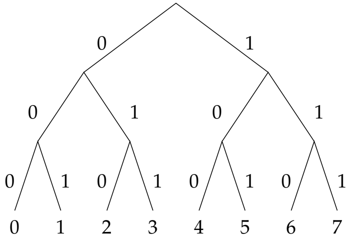
Dice(k)

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 $\{1, \cdots, 6\}$

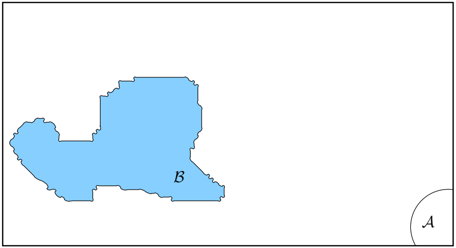
repeat
| $X =$**Dice-8()**
until $X \leqslant 6$
return X

Proof: later

Generation Methods based on Rejection

Principe

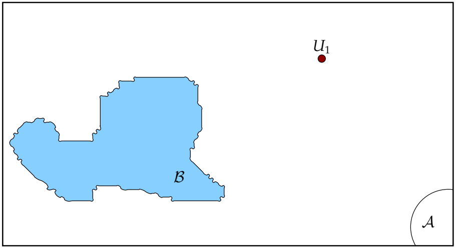
Generate uniformly on $\mathcal{A}$ accept if the point is in $\mathcal{B}$.



Generation Methods based on Rejection

Principe

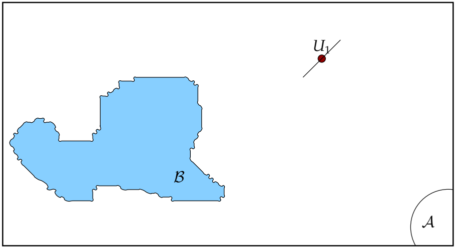
Generate uniformly on $\mathcal{A}$ accept if the point is in $\mathcal{B}$.



Generation Methods based on Rejection

Principe

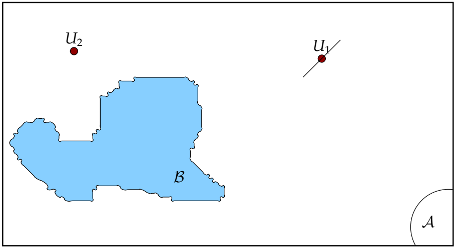
Generate uniformly on $\mathcal{A}$ accept if the point is in $\mathcal{B}$.



Generation Methods based on Rejection

Principe

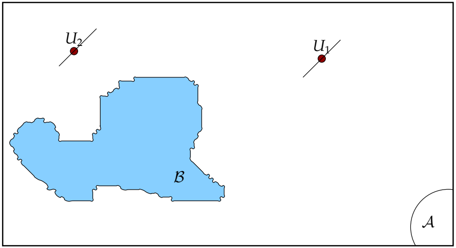
Generate uniformly on $\mathcal{A}$ accept if the point is in $\mathcal{B}$.



Generation Methods based on Rejection

Principe

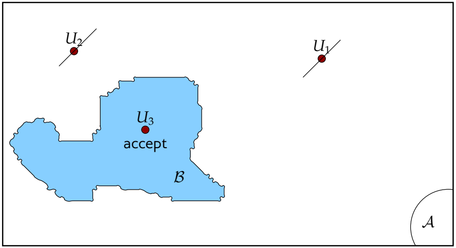
Generate uniformly on $\mathcal{A}$ accept if the point is in $\mathcal{B}$.



Generation Methods based on Rejection

Principe

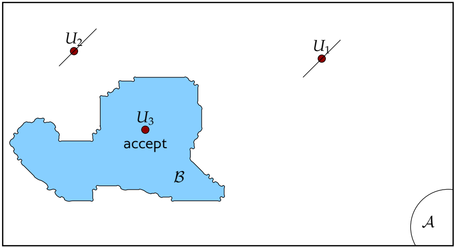
Generate uniformly on $\mathcal{A}$ accept if the point is in $\mathcal{B}$.



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($\mathcal{B}$)**: $X_1, X_2, \cdots, X_n, \cdots$

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

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

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 $p_a = \frac{|\mathcal{B}|}{|\mathcal{A}|}$ (probability of acceptance).
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 $[0,1)$

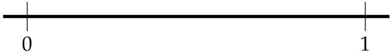
Probabilistic Model

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

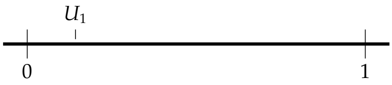
For all $n \in \mathbb{N}$, for all the intervals $[a_i, b_i)$ with $0 \leqslant i \leqslant n$ and $0 \leqslant a_i < b_i \leqslant 1$,

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

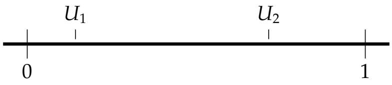
The random function



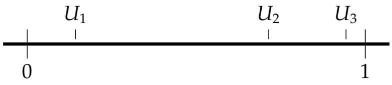
The random function



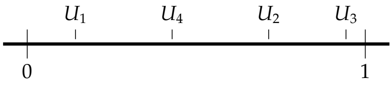
The random function



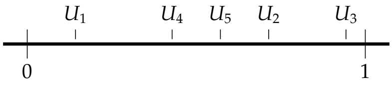
THE RANDOM FUNCTION



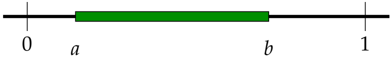
THE RANDOM FUNCTION



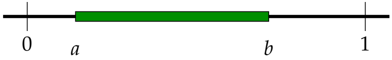
The random function



The random function

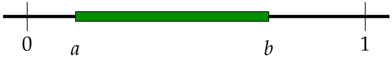


The random function



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

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 < 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, ··· , n}
  u=Random ()
  Return ⌈n * u⌉
  // smallest integer greater
     than u × n
```

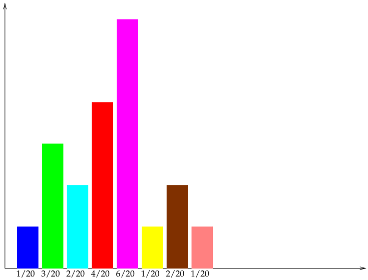
The problem

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 $i = 1, \quad i \leqslant n, \quad i++$
  for $j = 1, \quad j \leqslant m_i \quad j++$
    T[l]=i
    l++

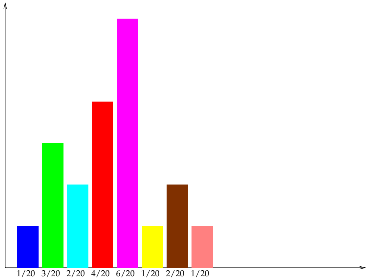
Generation

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

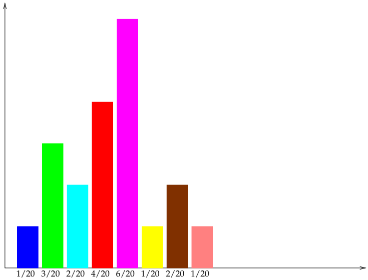
Generation algorithm

Generation (T)
Data: T Tabulation of distribution p
Result: A random number following distribution p
u=`Random` ()
$l = \lceil m * u \rceil$
`Return` $T[l]$

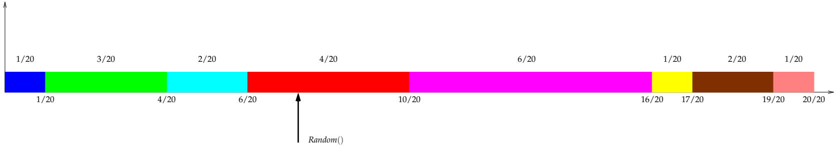
Probabilities on a Finite Set



PROBABILITIES ON A FINITE SET

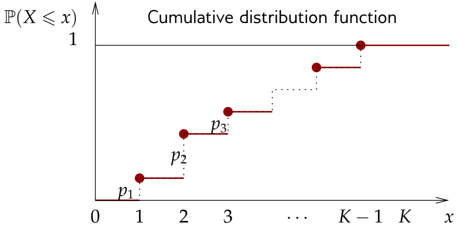


Histogram : Flat representation



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

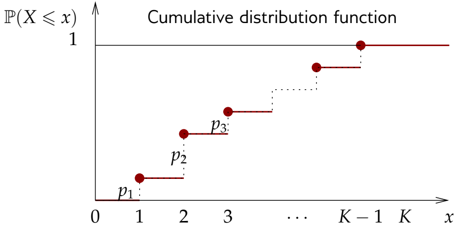
INVERSE OF PROBABILITY DISTRIBUTION FUNCTION



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

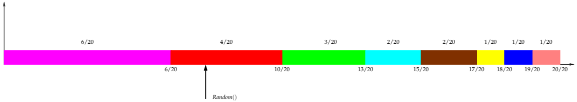
Generation (p)
Data: A distribution p
Result: A random number following distribution p

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

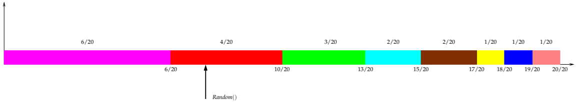


- use a dichotomy algorithm
- use a tree searching algorithm (optimality = Huffmann coding tree)

SEARCHING OPTIMIZATION

Optimization methods

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

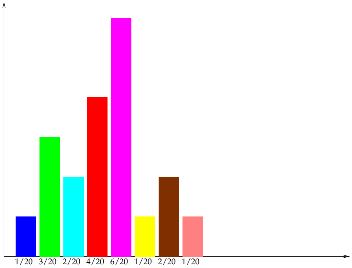


- use a dichotomy algorithm
- use a tree searching algorithm (optimality = Huffmann coding tree)

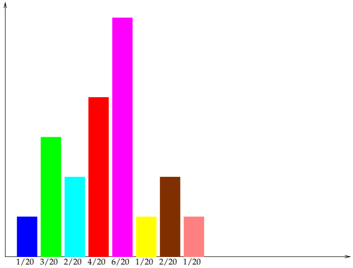
Comments

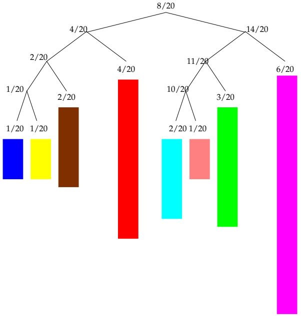
- Depends on the usage of the generator (repeated use or not)
- pre-computation usually $\mathcal{O}(K)$ could be huge
-

OPTIMALITY

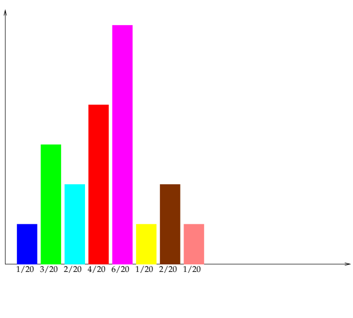


OPTIMALITY


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

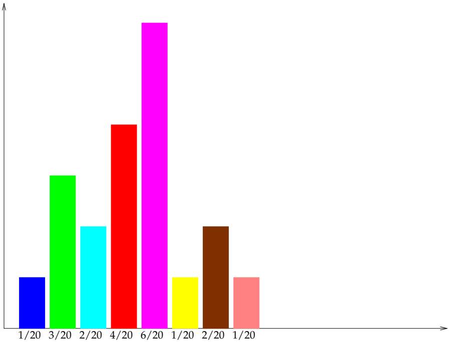


Number of comparisons

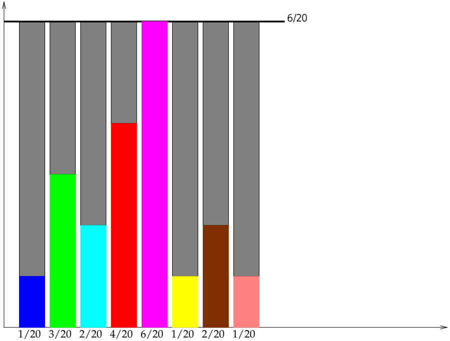
Binary Search Tree Structure

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

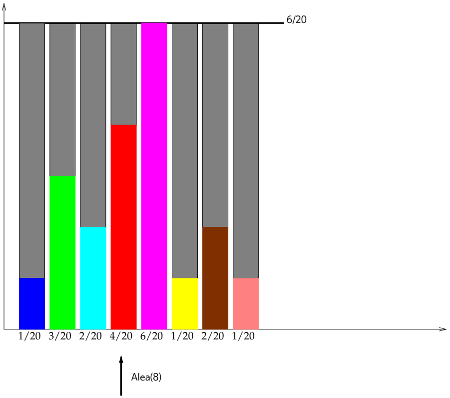
Rejection Based Methods



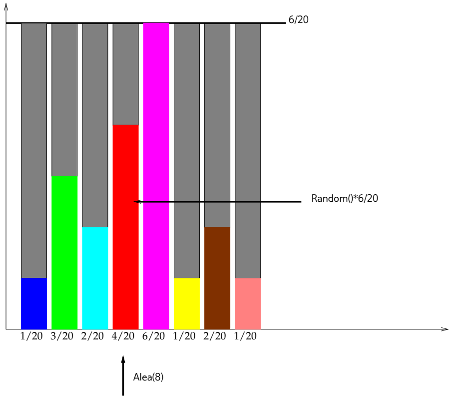
Rejection Based Methods



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 = ⌈n * u⌉
    v = Random() * p_max
    N++
until v ⩽ p_k
Return k, N
```

Proof

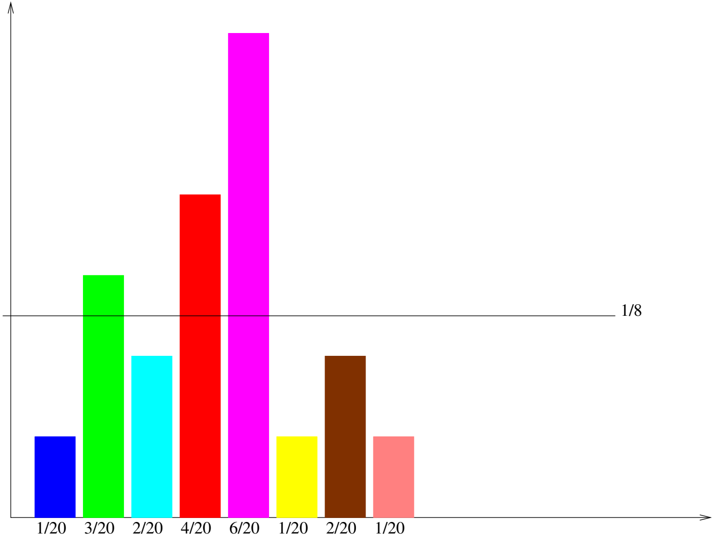
Same proof as for the uniform case

Complexity

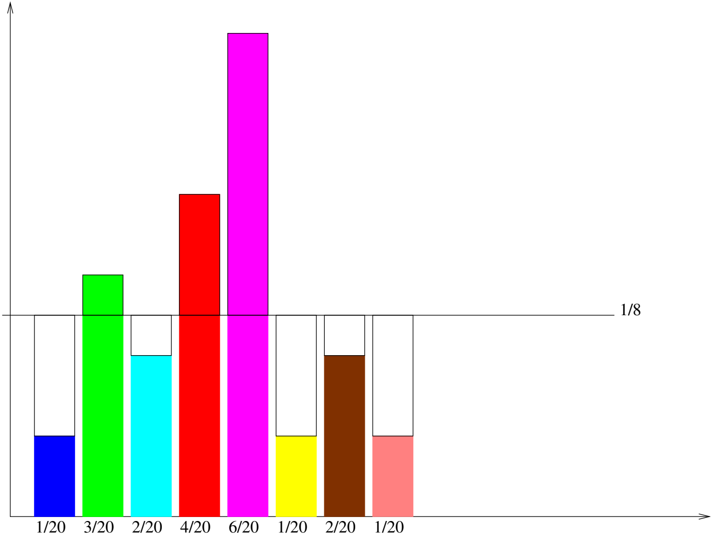
Average number of iterations :

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

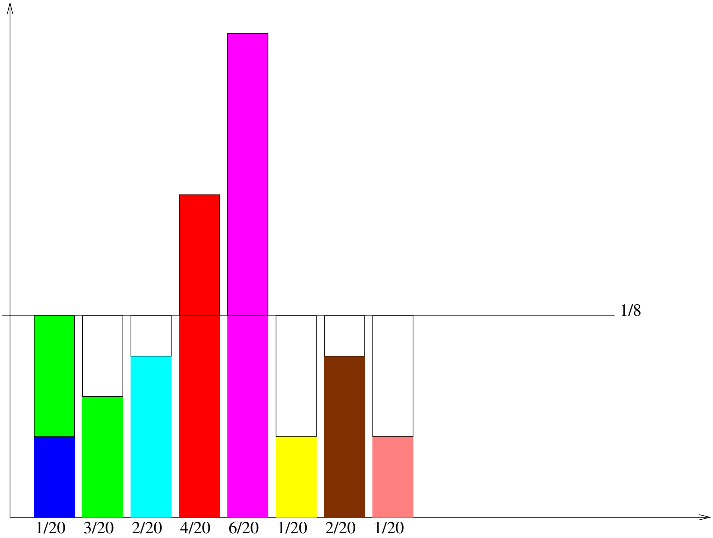
Aliasing Method



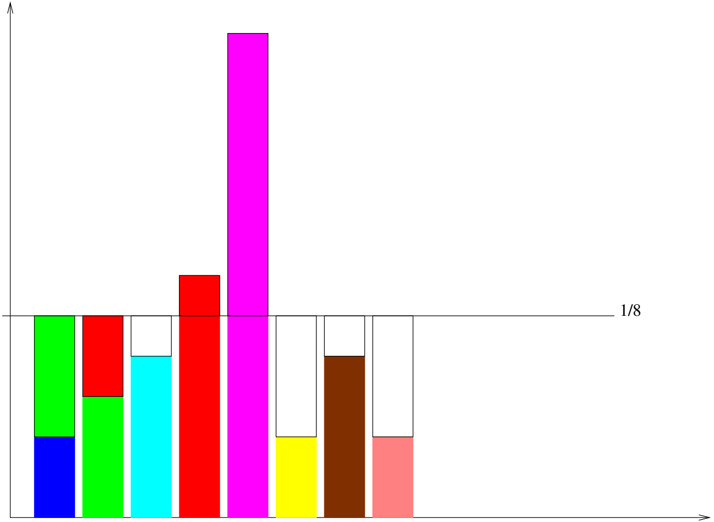
ALIASING METHOD



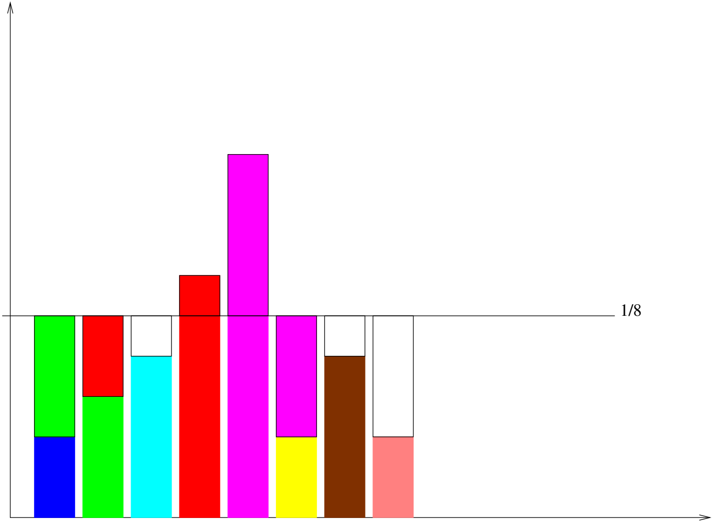
Aliasing Method



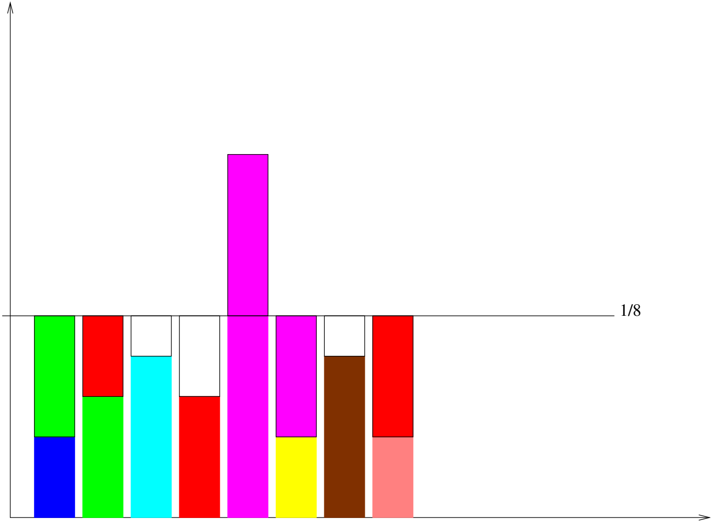
Aliasing Method



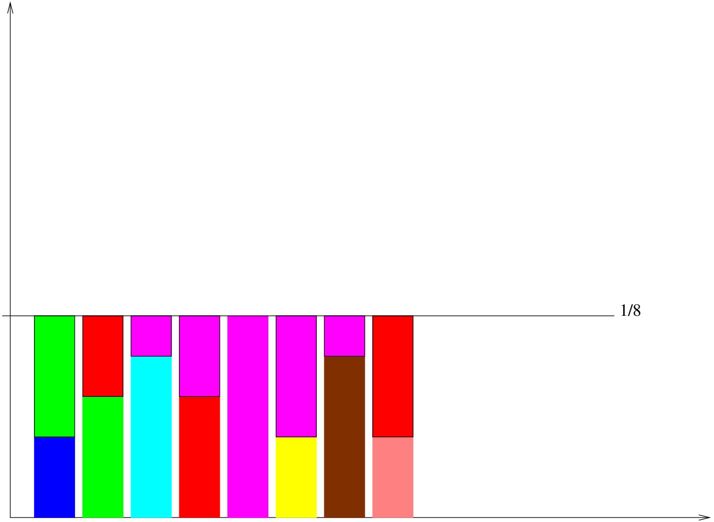
Aliasing Method



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 $p_k < \frac{1}{n}$ **do** $L = L \cup \{k\}$
    case $p_k > \frac{1}{n}$ **do** $U = U \cup \{k\}$
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 $< \frac{1}{n}$ **do** $L = L \cup \{k\}$
    case $> \frac{1}{n}$ **do** $U = U \cup \{k\}$

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 a distribution 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)

EXERCISES (1)

A typical law

Design at least 4 algorithms that simulate a pseudo-random variable according the empirical law :

c	A	B	C	D	E	F	G	H
$\mathbb{P}(X = c)$	0.10	0.20	0.05	0.05	0.05	0.15	0.35	0.05

Compute the average cost of the generation for each algorithm.

The power of 2

Design an algorithm that simulate a pseudo-random variable according the empirical law :

c	A	B	C	D	E	F	G	H	I	J
$\mathbb{P}(X = c)$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{8}$	$\frac{1}{32}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{8}$

Compute the average cost of the generation algorithm. What could you conclude ?

Application Exercise

On web servers it has been shown experimentally that hits on pages follow a Zipf's law. This law appears also in documents popularity in P2P systems, words occurrences in texts,...
Consider a web server with N pages. Pages are ranked by their popularity and let p_i be the probability of requesting page i. We have

$$p_1 \geqslant p_2 \geqslant \cdots \geqslant p_N$$

For the Zipf's law we have $p_i = \frac{1}{H_N}\frac{1}{i}$. This means that the second web page occurs approximately 1/2 as often as the first, and the third page 1/3 as often as the first, and so on.
H_N is the N^{th} harmonic number :

$$H_N = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{N}$$

which could be approximated by $\log N + \gamma + o(\frac{1}{N})$ with $\gamma = 0.5772156649$ the Euler constant.

- If N is small, classical techniques could be used. But what happens when N is large (10.000 or 100.000) ?
- Propose an algorithm that generates, approximatively, with the Zipf's law from a generator of real numbers on $[0, 1[$.
- Generalize this algorithm for “heavy-tail” laws (Benford's laws, Pareto's laws) with probability

  $$p_i = \frac{1}{H_{N,\alpha}}\frac{1}{i^\alpha},$$

  with α some “sharpness” coefficient and the normalization coefficient $H_{N,\alpha} = \left(\sum_1^N \frac{1}{i^\alpha}\right)^{-1}$.

Classical laws exercises

Binomial law

Propose several algorithms that simulate a variable X following the binomial distribution $\mathcal{B}in(n,k)$

$$\mathbb{P}(X=k) = \left(\begin{array}{c} n \\ k \end{array} \right) p^k (1-p)^{n-k}$$

Does the optimal method depends on the parameter values ?

Geometric distribution

Propose several algorithms that simulate a variable following the geometric distribution $\mathcal{G}(p)$

$$\mathbb{P}(X=k) = (1-p)p^{k-1}$$

Does the optimal method depends on the parameter values ?

Poisson distribution

Propose several algorithms that simulate a variable following the Poisson distribution $\mathcal{P}(\lambda)$

$$\mathbb{P}(X=k) = e^{-\lambda}\frac{\lambda^k}{k!}$$

Does the optimal method depends on the parameter values ?

Generation of complex configurations

Examples

- sequences of requests on a web server
- path in a graph
- interconnexion graph
- memory configuration
- mine field
- ...

Mine field

Write an algorithm that generates a random mine field with exactly $k(= 10)$ mines in a n field. Example for $n = 9 \times 9$

Mine field (2)

Uniform generation of a mine field with exactly exactement k mines

Method 1 : There are exactly $\binom{n}{k}$ different mine field, number them, generate uniformly an index of a mine field in $\{1, \cdots, \binom{n}{k}\}$

Method 2 : Generate uniformly a permutation of $\{1, \cdots, n\}$ and take the first k elements as mine positions

Method 2bis : Generate in sequence uniformly the mines on the available positions.

Method 3 : While the number of mines is not sufficient pick uniformly a position in $\{1, ..., n\}$ and put a mine if the position is free

Method 4 : We put successively a mine in position i with probability $\frac{k-k_i}{n-i+1}$, where k_i is the number of mines in positions $\{1, \cdots, i-1\}$.

Generation of mean field with average density $d = \frac{k}{n}$ de mines

Method 5 : Flip a biaised coin with probability d in each position to put mines.

Mehode 5b : Same method but reject the mine field if the average density is out of $[d - \epsilon, d + \epsilon]$.

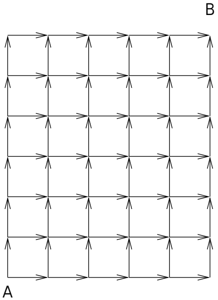
Paths generation

In a "feed-forward" communication network generate uniformly a route between 2 given nodes

Paths generation

In a "feed-forward" communication network generate uniformly a route between 2 given nodes

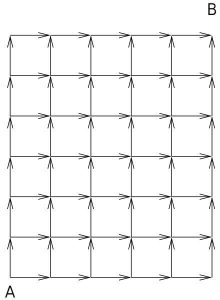
Manhattan Topology



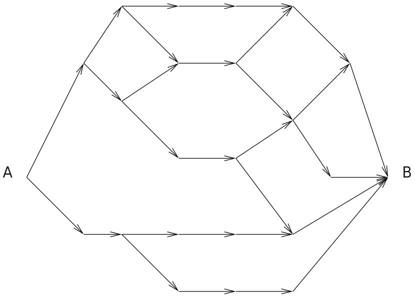
Paths generation

In a "feed-forward" communication network generate uniformly a route between 2 given nodes

Manhattan Topology



General topology



Graph generation

Typical graph

Generate a random graph uniformly (directed or non-directed)

- without constraints
- with a given number of edges
- with a fixed degree
- connected
- imagine your own constraints

DOMINOES

The dominoes game is a set of all the tiles marked by 2 marks, these marks are in $\{0, \cdots, n\}$. Then a domino is defined by a couple (i, j) with $0 \leqslant i \leqslant j \leqslant 6$.

- **Number of dominoes :** Compute K_6 the number of tiles of a classical game with $n = 6$. Deduce K_n of a game with marks between 0 and n
- **Generator of dominoes** Write an algorithm that fe-generates uniformly a dominoe for a given n
- **Cost of the generation** Compute the complexity of the generation including pre-computation if ever

Generation of binary research tree

Generation of binary research tree

Uniform recursive decomposition

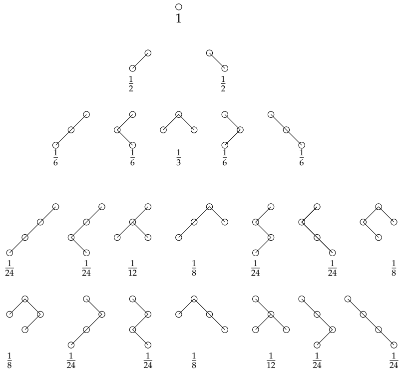
```
Random_BST (n)
  Data: n number of nodes
  Result: a random tree
  if n = 0
    return empty_tree ()
  else
    q = Random (0, n − 1)
    A1 = Random_BST (q)
    A2 = Random_BST (n − 1 − q)
    return join (A1, A2)
```

GENERATION OF BINARY RESEARCH TREE

Uniform recursive decomposition

```
Random_BST (n)
  Data: n number of nodes
  Result: a random tree
  if n = 0
    return empty_tree ()
  else
    q = Random (0, n − 1)
    A1 = Random_BST (q)
    A2 = Random_BST (n − 1 − q)
    return join (A1, A2)
```

Non uniform on binary trees



Uniform Generation of Binary Trees

Catalan's Numbers

Recurrence equation

$$C_0 = C_1 = 1;$$

$$C_N = \sum_{q=0}^{n-1} C_q C_{n-1-q}.$$

Then

$$1 = \sum_{q=0}^{n-1} \frac{C_q C_{n-1-q}}{C_n} = \sum_{q=0}^{n-1} p_{n,q}.$$

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

Uniform Generation of Binary Trees

Catalan's Numbers

Recurrence equation

$$C_0 = C_1 = 1;$$

$$C_N = \sum_{q=0}^{n-1} C_q C_{n-1-q}.$$

Then

$$1 = \sum_{q=0}^{n-1} \frac{C_q C_{n-1-q}}{C_n} = \sum_{q=0}^{n-1} p_{n,q}.$$

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

Génération uniforme

```
Random_BT (n)
  Data: n number of nodes
  Result: a random tree
  if n = 0
    return empty_tree ()
  else
    q=Generate(p_{n,0}, ··· , p_{n,n-1})
    A_1 =Random_BT (q)
    A_2 =Random_BT (n - 1 - q)
    return join (A_1, A_2)
```

pre-computation $p_{n,q}$

$$p_{n,q} = \frac{C_q C_{n-1-q}}{C_n}.$$

Labelled trees

How many labelled trees with n nodes ?
Propose an algorithm that generates uniformly random trees.

Labelled trees

How many labelled trees with n nodes ?
Propose an algorithm that generates uniformly random trees.
Cayley's formulae

$$T_n = n^{n-2}.$$

Prüfer's coding algorithm.

SYNTHESIS

Simulation is a powerful tool for computation (randomized algorithms)

- Probabilistic specification based on statistical properties (uniformity, independence, goodness of fit,...)
- Proof of statistical properties
- Complexity (probabilistic), average computation time
- Complex objects : link between combinatorial decomposition and simulation algorithm

Based on a `Random` function (external)

- Primality testing (security)
- Time randomization (networking)
- Monte-Carlo method (scientific computations)
- Test covering (verification)
- Statistical learning (Bayesian approach)
- Simulated Annealing (optimization, NP-complete problems)