# Performance Evaluation : Probabilistic simulation

## Stochastic Modeling of Computer Systems
## MOSIG Master 2

Jean-Marc Vincent

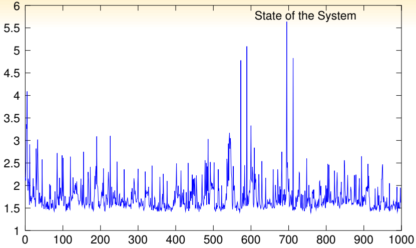
Laboratoire LIG, projet Inria-Mescal
UniversitéJoseph Fourier
Jean-Marc.Vincent@imag.fr

2012 November 5

# Outline

1. **Motivation**
   - Convergence
   - Solving
   - Simulation
2. Discrete generation
3. Perfect sampling
4. Case Studies

# Long Run Evolution and Time Scaling



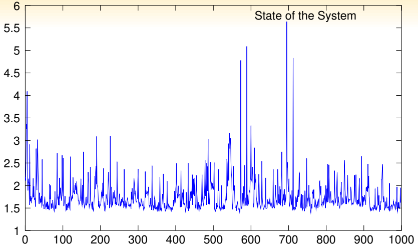

**Typical trajectory**

Performance of the system ⇒ analysis of the steady-state

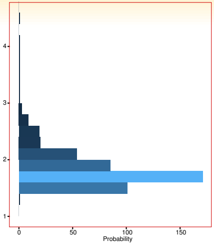
Computation of the steady-state

**Main contribution**
Efficient computation in finite time of stationary samples

# Long Run Evolution and Time Scaling




Typical trajectory




Statistical Synthesis

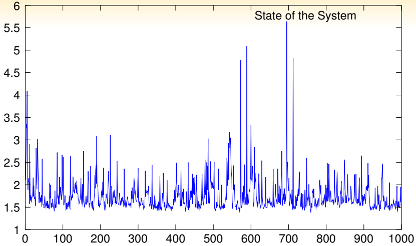
**Steady-State**

Performance of the system $\Rightarrow$ analysis of the steady-state

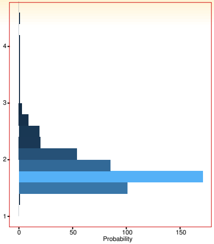
Computation of the steady-state

**Main contribution**
Efficient computation in finite time of stationary samples

# Long Run Evolution and Time Scaling




**Typical trajectory**




**Statistical Synthesis**

**Steady-State**

Performance of the system $\Rightarrow$ analysis of the steady-state

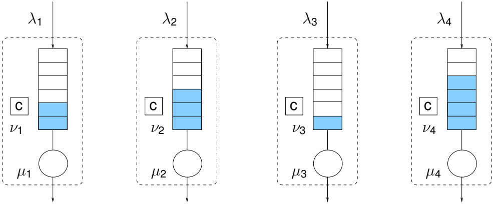
Computation of the steady-state

**Main contribution**

Efficient computation in finite time of stationary samples

# Load sharing model

Parallel independent queues




**State space**: number of tasks in each queue; $\mathcal{X}_1 \times \cdots \times \mathcal{X}_K$
**Dynamics**: events driven by Poisson process

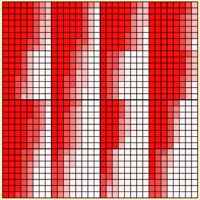
- Generation of a new task in a queue, with rate $\lambda$
- Task completion, with rate $\mu$
- Control, with rate $\nu$

**Uniformization** $\Rightarrow$ Stochastic Recurence Equation $X_{n+1} = \Phi(X_n, E_{n+1})$

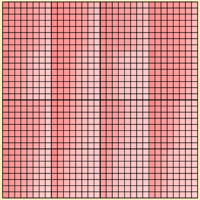
# Application

## Push on arrival

Input Load

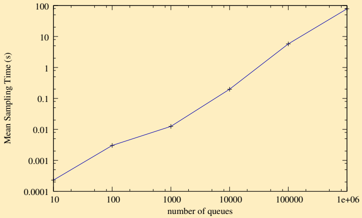


Hierarchical Load Sharing



## Scaling Toward million of nodes

Policy: Threshold Push on Arrival with priority list of 8 nodes

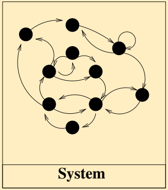



The time to simulate such system is linear with the number of nodes

[ASMTA 2010]

# Modeling and Analysis of Computer Systems

## Complex system




## Basic model assumptions

System :
- automaton (discrete state space)
- **discrete** or continuous time

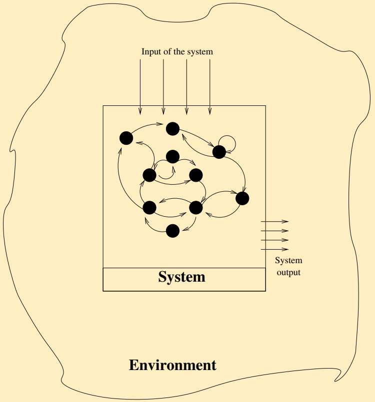
Environment : non deterministic
- time homogeneous
- stochastically regular

## Problem

**Understand "typical" states**
- steady-state estimation
- ergodic simulation
- state space exploring techniques

# Modeling and Analysis of Computer Systems

## Complex system




## Basic model assumptions

System :
- automaton (discrete state space)
- **discrete** or continuous time

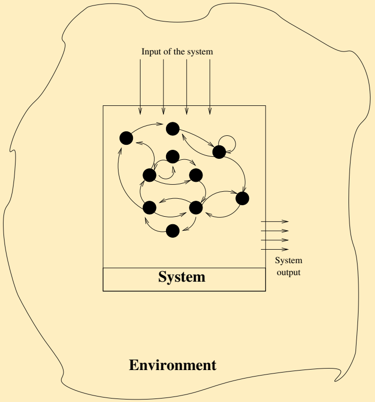
Environment : non deterministic
- time homogeneous
- stochastically regular

## Problem

**Understand "typical" states**
- steady-state estimation
- ergodic simulation
- state space exploring techniques

# Modeling and Analysis of Computer Systems

## Complex system




## Basic model assumptions

System :
- automaton (discrete state space)
- **discrete** or continuous time

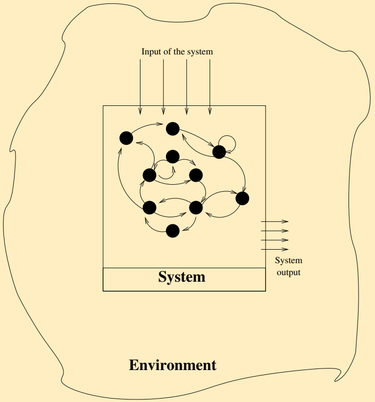
Environment : non deterministic
- time homogeneous
- stochastically regular

## Problem

**Understand "typical" states**
- steady-state estimation
- ergodic simulation
- state space exploring techniques

# Modeling and Analysis of Computer Systems

## Complex system




## Basic model assumptions

System :
- automaton (discrete state space)
- **discrete** or continuous time

Environment : non deterministic
- time homogeneous
- stochastically regular

## Problem

**Understand "typical" states**
- steady-state estimation
- ergodic simulation
- state space exploring techniques

# Convergence In Law

Let $\{X_n\}_{n\in\mathbb{N}}$ a homogeneous, irreducible and aperiodic Markov chain taking values in a discrete state $\mathcal{X}$ then

- The following limits exist (and do not depend on $i$)

$$\lim_{n\to+\infty} \mathbb{P}(X_n = j|X_0 = i) = \pi_j;$$

- $\pi$ is the unique probability vector invariant by $P$

$$\pi P = \pi;$$

- The convergence is rapid (geometric); there is $C > 0$ and $0 < \alpha < 1$ such that

$$||\mathbb{P}(X_n = j|X_0 = i) - \pi_j|| \leqslant C.\alpha^n.$$

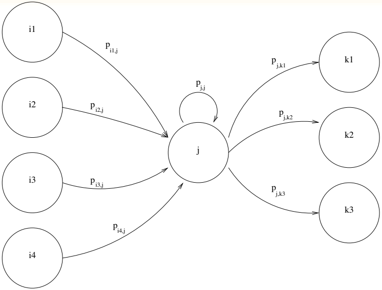
Denote

$$X_n \xrightarrow{\mathcal{L}} X_\infty;$$

with $X_\infty$ with law $\pi$
$\pi$ is the **steady-state probability** associated to the chain

# Interpretation

## Equilibrium equation



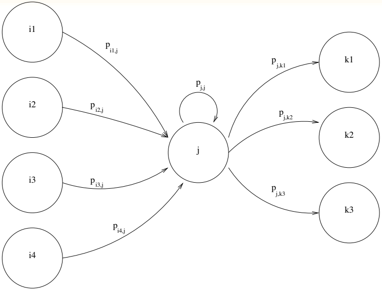

Probability to enter $j$ =probability to exit $j$
**balance equation**

$$\sum_{i \neq j} \pi_i p_{i,j} = \sum_{k \neq j} \pi_j p_{j,k} = \pi_j \sum_{k \neq j} p_{j,k} = \pi_j (1 - p_{j,j})$$

$\pi \stackrel{def}{=}$ steady-state.
If $\pi_0 = \pi$ the process is stationary ($\pi_n = \pi$)

# Interpretation

## Equilibrium equation



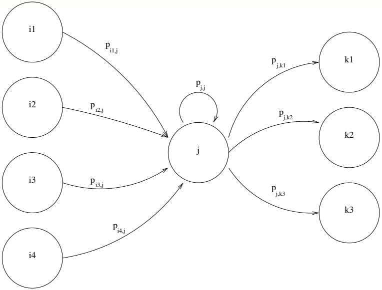

Probability to enter $j$ =probability to exit $j$
**balance equation**

$$\sum_{i \neq j} \pi_i p_{i,j} = \sum_{k \neq j} \pi_j p_{j,k} = \pi_j \sum_{k \neq j} p_{j,k} = \pi_j (1 - p_{j,j})$$

$\pi \stackrel{def}{=}$ steady-state.
If $\pi_0 = \pi$ the process is stationary ($\pi_n = \pi$)

# Interpretation

## Equilibrium equation




Probability to enter $j$ =probability to exit $j$
**balance equation**

$$\sum_{i \neq j} \pi_i p_{i,j} = \sum_{k \neq j} \pi_j p_{j,k} = \pi_j \sum_{k \neq j} p_{j,k} = \pi_j (1 - p_{j,j})$$

$\pi \stackrel{def}{=}$ steady-state.
If $\pi_0 = \pi$ the process is stationary ($\pi_n = \pi$)

# Ergodic Theorem

Let $\{X_n\}_{n\in\mathbb{N}}$ a homogeneous aperiodic and irreducible Markov chain on $\mathcal{X}$ with steady-state probability $\pi$ then

- for all function $f$ satisfying $\mathbb{E}_\pi|f| < +\infty$

$$\frac{1}{N}\sum_{n=1}^{N} f(X_n) \xrightarrow{P-p.s.} \mathbb{E}_\pi f.$$

generalization of the *strong law of large numbers*

- If $\mathbb{E}_\pi f = 0$ then there exist $\sigma$ such that

$$\frac{1}{\sigma\sqrt{N}}\sum_{n=1}^{N} f(X_n) \xrightarrow{\mathcal{L}} \mathcal{N}(0,1).$$

generalization of the *central limit theorem*

# Fundamental question

**Given a function $f$ (cost, reward, performance,...) estimate**

$$\mathbb{E}_\pi f$$

**and give the quality of this estimation.**

# Solving methods

## Solving $\pi = \pi P$

- Analytical/approximation methods
- Formal methods $N \leqslant 50$
  Maple, Sage,...
- Direct numerical methods $N \leqslant 1000$
  Mathematica, Scilab,...
- Iterative methods with preconditioning $N \leqslant 100,000$
  Marca,...
- Adapted methods (structured Markov chains) $N \leqslant 1,000,000$
  PEPS,...
- Monte-Carlo simulation $N \geqslant 10^7$

## Postprocessing of the stationary distribution

Computation of rewards (expected stationary functions)
Utilization, response time,...

# Ergodic Sampling(1)

## Ergodic sampling algorithm

Representation : **transition fonction**

$$X_{n+1} = \Phi(X_n, e_{n+1}).$$

$x \leftarrow x_0$
{choice of the initial state at time =0}
$n = 0;$
**repeat**
  $n \leftarrow n + 1;$
  $e \leftarrow Random\_event();$
  $x \leftarrow \Phi(x, e);$
  Store $x$
  {computation of the next state $X_{n+1}$}
**until** some empirical criteria
return the trajectory

**Problem : Stopping criteria**

# Ergodic Sampling(2)

## Start-up

Convergence to stationary behavior

$$\lim_{n \to +\infty} \mathbb{P}(X_n = x) = \pi_x.$$

**Warm-up period :** Avoid initial state dependence
Estimation error :

$$||\mathbb{P}(X_n = x) - \pi_x|| \leqslant C\lambda_2^n.$$

$\lambda_2$ second greatest eigenvalue of the transition matrix
- bounds on $C$ and $\lambda_2$ (spectral gap)
- cut-off phenomena

$\lambda_2$ and $C$ non reachable in practice
(complexity equivalent to the computation of $\pi$)
some known results (Birth and Death processes)

# Ergodic Sampling(3)

## Estimation quality

Ergodic theorem :

$$\lim_{n \to +\infty} \frac{1}{n} \sum_{i=1}^{n} f(X_i) = \mathbb{E}_\pi f.$$

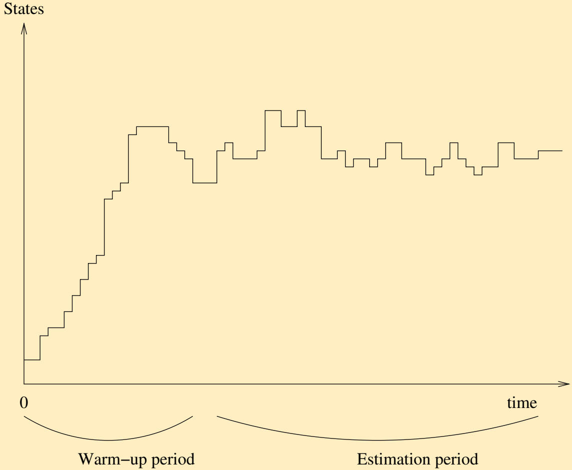
**Length of the sampling :** Error control (CLT theorem)

## Complexity

Complexity of the transition function evaluation (computation of $\Phi(x, .)$)
Related to the stabilization period + Estimation time

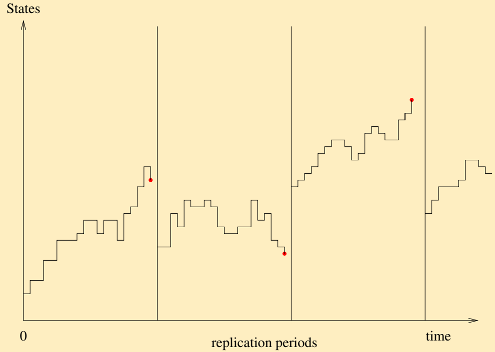
# Ergodic sampling(4)

## Typical trajectory



# Replication Method

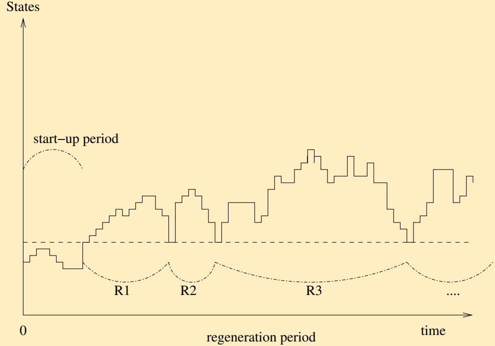
## Typical trajectory




Sample of independent states
Drawback : length of the replication period (dependence from initial state)

# Regeneration Method

## Typical trajectory




Sample of independent trajectories
Drawback : length of the regeneration period (choice of the regenerative state)

# Outline

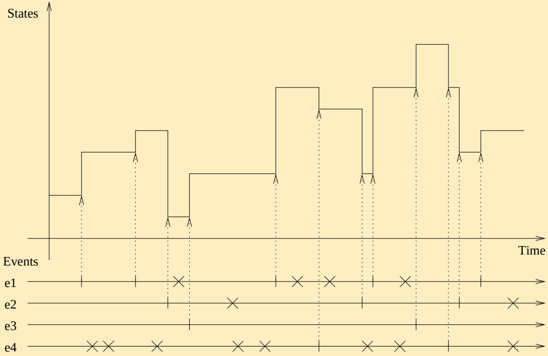
# Event Modelling

Multidimensional state space : $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_K$ with $\mathcal{X}_i = \{0, \cdots, C_i\}$.

Event $e$ :

$\rightsquigarrow$ transition function $\Phi(., e)$; (skip rule)

$\rightsquigarrow$ Poisson process $\lambda_e$




1781-1840

# Event modelling

## Uniformization

$$\Lambda = \sum_e \lambda_e \text{ and } \mathbb{P}(event\ e) = \frac{\lambda_e}{\Lambda};$$

Trajectory : $\{e_n\}_{n\in\mathbb{Z}}$ i.i.d. sequence.
$\Rightarrow$ Homogeneous Discrete Time Markov Chain [Bremaud 99]
$X_{n+1} = \Phi(X_n, e_{n+1})$.

Generation among a small finite space $\mathcal{E}$ : $\mathcal{O}(1)$

# Generating random objects

Denote by $X$ the generated object ( $X$ is a random variable)
Distribution (proportion of observations, input of the load injector)

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

Remarks :

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

Expectation (average, mean)

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

Variance and standard deviation

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

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

# Generating random objects

Denote by $X$ the generated object ( $X$ is a random variable)
Distribution (proportion of observations, input of the load injector)

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

Remarks :

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

Expectation (average, mean)

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

Variance and standard deviation

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

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

# Generating random objects

Denote by $X$ the generated object ( $X$ is a random variable)
Distribution (proportion of observations, input of the load injector)

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

Remarks :

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

Expectation (average, mean)

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

Variance and standard deviation

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

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

## Generating random objects

Denote by $X$ the generated object ( $X$ is a random variable)
Distribution (proportion of observations, input of the load injector)

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

Remarks :

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

Expectation (average, mean)

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

Variance and standard deviation

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

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

# The random function

Random bit generator (see previous lecture)

**drand48 manpage**

double drand48(void) (48 bits encoded in 8 bytes)
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 independent random variables uniformly distributed on the real interval $[0, 1[$.**

# The random function

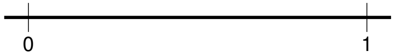
Random bit generator (see previous lecture)

**drand48 manpage**

double drand48(void) (48 bits encoded in 8 bytes)
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 independent random variables uniformly distributed on the real interval $[0, 1[$.**

# The random function



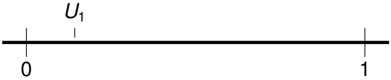

**Problem**

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

**Example : flip a coin**

u= r andom()
**if** u $\leqslant \frac{1}{2}$ **then**
    return Head
**else**
    return Tail
**end if**

# The random function



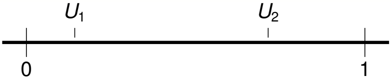

**Problem**

All the difficulty is to find a function (an algorithm) that transforms the [0, 1[ in a set with a good probability conserving.

**Example : flip a coin**

u= random()
**if** u $\leqslant \frac{1}{2}$ **then**
  return Head
**else**
  return Tail
**end if**

# The random function



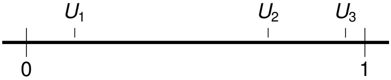

**Problem**

All the difficulty is to find a function (an algorithm) that transforms the [0, 1[ in a set with a good probability conserving.

**Example : flip a coin**

```
u= r andom()
if u ≤ 1/2 then
   return Head
else
   return Tail
end if
```

# The random function



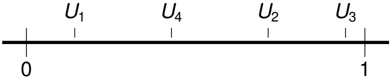

**Problem**

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

**Example : flip a coin**

u= r andom()
**if** u $\leqslant \frac{1}{2}$ **then**
  return Head
**else**
  return Tail
**end if**

# The random function



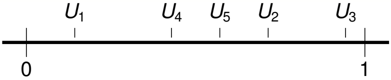

**Problem**

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

**Example : flip a coin**

u= r andom()
**if** u $\leqslant \frac{1}{2}$ **then**
  return Head
**else**
  return Tail
**end if**

# The random function




**Problem**

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

**Example : flip a coin**

u= r andom()
**if** u $\leq \frac{1}{2}$ **then**
  return Head
**else**
  return Tail
**end if**

# The random function

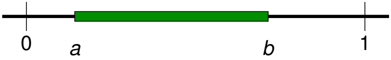

**Problem**

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

**Example : flip a coin**

u= r andom()
**if** u $\leqslant \frac{1}{2}$ **then**
  return Head
**else**
  return Tail
**end if**

# The random function




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

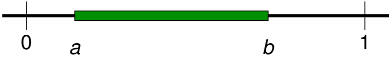
### Problem

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

### Example : flip a coin

```
u= r andom()
if u ⩽ 1/2 then
   return Head
else
   return Tail
end if
```

# The random function




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

## Problem

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

## Example : flip a coin

u= random()
**if** u $\leqslant \frac{1}{2}$ **then**
  return Head
**else**
  return Tail
**end if**

# The random function


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

## Problem

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

## Example : flip a coin

u= r andom()
**if** u $\leqslant \frac{1}{2}$ **then**
    return Head
**else**
    return Tail
**end if**

# Practical example : Web server

## Types of request

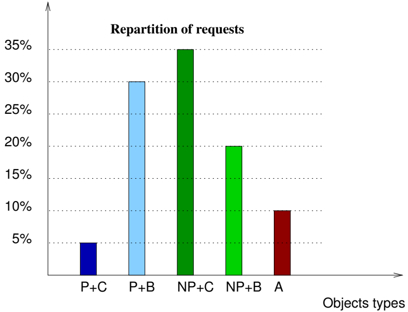
1. Professional customer, consult
2. Professional customer, purchase
3. Non professional customer, consult
4. Non professional customer, purchase
5. Adminstration

Build an algorithm that provides a set of requests according the observed distribution.

# Practical example : Web server

## Types of request

1. Professional customer, consult
2. Professional customer, purchase
3. Non professional customer, consult
4. Non professional customer, purchase
5. Adminstration

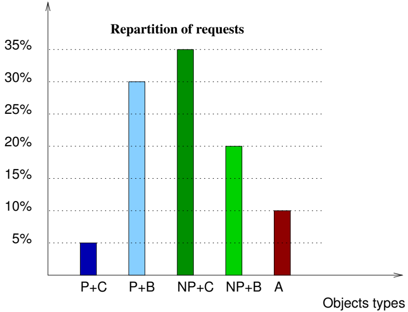



Build an algorithm that provides a set of requests according the observed distribution.

# Practical example : Web server

## Types of request

1. Professional customer, consult
2. Professional customer, purchase
3. Non professional customer, consult
4. Non professional customer, purchase
5. Adminstration




Build an algorithm that provides a set of requests according the observed distribution.

# 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

i=0
**for** k=1, k $\leqslant$ K, k++ **do**
  **for** j=1, j $\leqslant$ $m_k$, j++ **do**
    T[i]= k; i=i+1;
  **end for**
**end for**

## Generation

Generate uniformly on the set $\{0, 1, \cdots, m-1\}$
Returns the value in the table
Computation cost : $\mathcal{O}(1)$ step
Memory cost : $\mathcal{O}(m)$

## Generation algorithm

u= r andom();
i= (int) floor(u*m)
**return** T[i]

# 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

i=0
**for** k=1, k $\leqslant$ K, k++ **do**
  **for** j=1, j $\leqslant$ $m_k$, j++ **do**
    T[i]= k; i=i+1;
  **end for**
**end for**

## Generation

Generate uniformly on the set $\{0, 1, \cdots, m-1\}$
Returns the value in the table
Computation cost : $\mathcal{O}(1)$ step
Memory cost : $\mathcal{O}(m)$

## Generation algorithm

u= r andom();
i= (int) floor(u*m)
**return** T[i]

# 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

i=0
**for** k=1, k $\leqslant$ K, k++ **do**
  **for** j=1, j $\leqslant$ $m_k$, j++ **do**
    T[i]= k; i=i+1;
  **end for**
**end for**

## Generation

Generate uniformly on the set $\{0, 1, \cdots, m-1\}$
Returns the value in the table
Computation cost : $\mathcal{O}(1)$ step
Memory cost : $\mathcal{O}(m)$

## Generation algorithm

u= r andom();
i= (int) floor(u*m)
**return** T[i]

# 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

```
i=0
for k=1, k ⩽ K, k++ do
   for j=1, j ⩽ m_k, j++ do
      T[i]= k; i=i+1;
   end for
end for
```

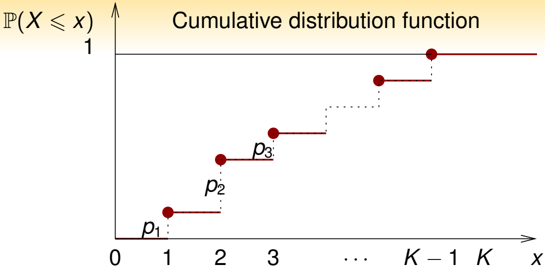
## Generation

Generate uniformly on the set
$\{0, 1, \cdots, m-1\}$
Returns the value in the table
Computation cost : $\mathcal{O}(1)$ step
Memory cost : $\mathcal{O}(m)$

## Generation algorithm

```
u= r andom();
i= (int) floor(u*m)
return T[i]
```

# Inverse of PDF



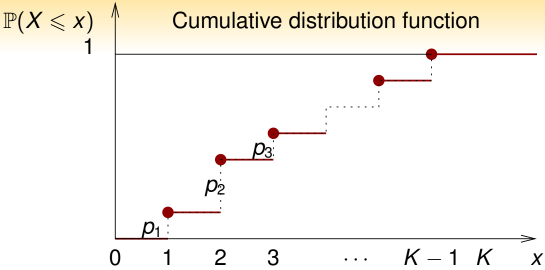

## Generation

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

```
s=0; k=0;
u=random()
while u >s do
   k=k+1
   s=s+p_k
end while
return k
```

# Inverse of PDF




## 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

s=0; k=0;
u=random()
**while** u $>$s **do**
  k=k+1
  s=s+$p_k$
**end while**
**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)

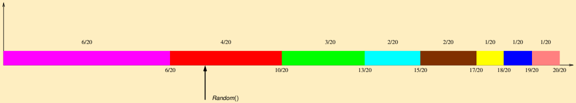
## Comments

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

# 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)

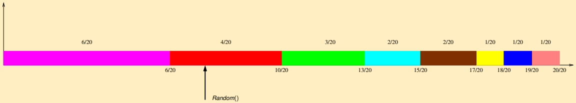
## Comments

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

# Searching optimization

## Optimization methods

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

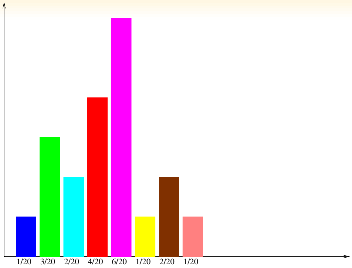



- use a dichotomy algorithm
- use a tree searching algorithm (optimality = Huffmann coding tree)

## Comments

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

# Optimality

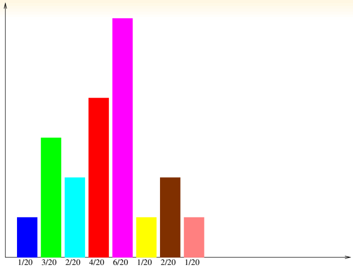


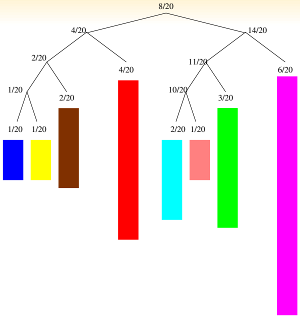

## Number of comparisons

Binary search tree structure

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

# Optimality



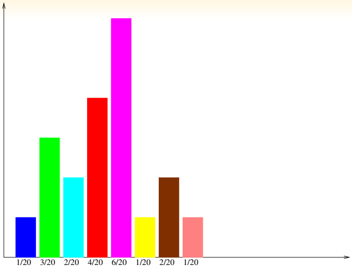



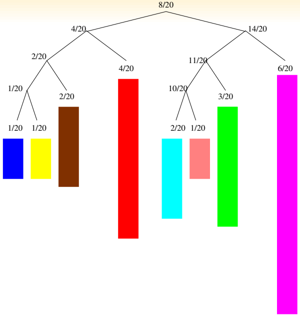

## Number of comparisons

Binary search tree structure

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

# Optimality




## Number of comparisons

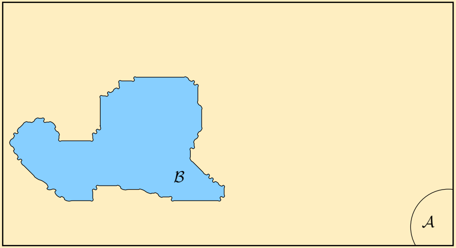
Binary search tree structure

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

# Rejection technique

## Base of the method

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




## Rejection algorithm

**repeat**
  x = uniform-generate($\mathcal{A}$)
**until** x$\in \mathcal{B}$
**return** x

## Complexity

Acceptance probability

$$p_a = \frac{Size(\mathcal{B})}{Size(\mathcal{A})}$$

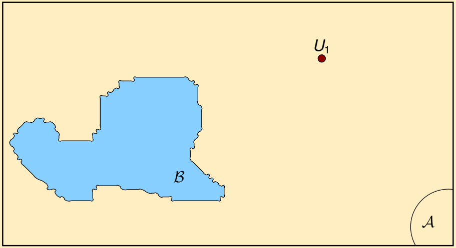
$N$ number of iterations

$$\mathbb{E}N = \frac{1}{p_a}.$$

# Rejection technique

### Base of the method

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




### Rejection algorithm

**repeat**
  x = uniform-generate($\mathcal{A}$)
**until** x$\in \mathcal{B}$
**return** x

### Complexity

Acceptance probability

$$p_a = \frac{Size(\mathcal{B})}{Size(\mathcal{A})}$$

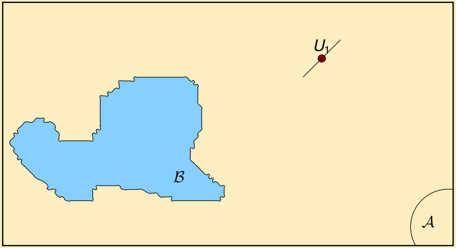
$N$ number of iterations

$$\mathbb{E}N = \frac{1}{p_a}.$$

# Rejection technique

## Base of the method

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




## Rejection algorithm

**repeat**
  x = uniform-generate($\mathcal{A}$)
**until** x$\in$ $\mathcal{B}$
**return** x

## Complexity

Acceptance probability

$$p_a = \frac{Size(\mathcal{B})}{Size(\mathcal{A})}$$

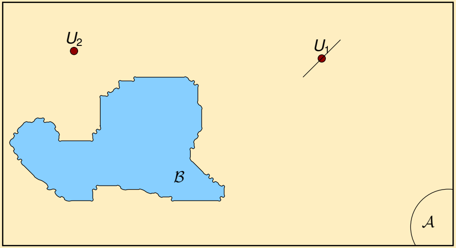
$N$ number of iterations

$$\mathbb{E}N = \frac{1}{p_a}.$$

# Rejection technique

## Base of the method

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




## Rejection algorithm

**repeat**
  x = uniform-generate($\mathcal{A}$)
**until** x$\in$ $\mathcal{B}$
**return** x

## Complexity

Acceptance probability

$$p_a = \frac{Size(\mathcal{B})}{Size(\mathcal{A})}$$

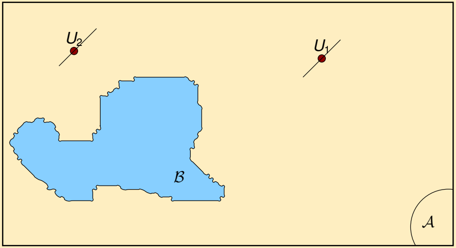
$N$ number of iterations

$$\mathbb{E}N = \frac{1}{p_a}.$$

# Rejection technique

## Base of the method

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




## Rejection algorithm

**repeat**
  x = uniform-generate($\mathcal{A}$)
**until** x$\in \mathcal{B}$
**return** x

## Complexity

Acceptance probability

$$p_a = \frac{Size(\mathcal{B})}{Size(\mathcal{A})}$$

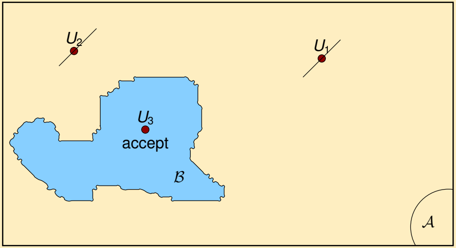
$N$ number of iterations

$$\mathbb{E}N = \frac{1}{p_a}.$$

# Rejection technique

## Base of the method

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




## Rejection algorithm

**repeat**
    x = uniform-generate($\mathcal{A}$)
**until** x$\in$ $\mathcal{B}$
**return** x

## Complexity

Acceptance probability

$$p_a = \frac{Size(\mathcal{B})}{Size(\mathcal{A})}$$

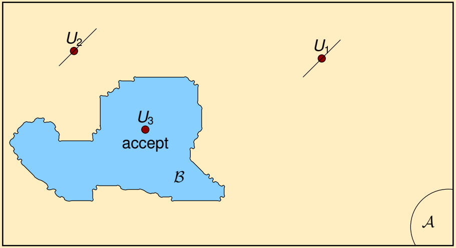
$N$ number of iterations

$$\mathbb{E}N = \frac{1}{p_a}.$$

# Rejection technique

## Base of the method

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




## Rejection algorithm

**repeat**
  x = uniform-generate($\mathcal{A}$)
**until** x$\in \mathcal{B}$
**return** x

## Complexity

Acceptance probability

$$p_a = \frac{Size(\mathcal{B})}{Size(\mathcal{A})}$$

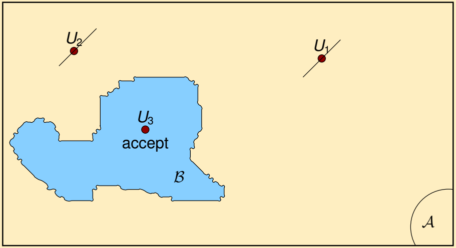
$N$ number of iterations

$$\mathbb{E}N = \frac{1}{p_a}.$$

# Rejection technique

## Base of the method

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




## Rejection algorithm

**repeat**
  x = uniform-generate($\mathcal{A}$)
**until** x$\in \mathcal{B}$
**return** x

## Complexity

Acceptance probability

$$p_a = \frac{Size(\mathcal{B})}{Size(\mathcal{A})}$$

$N$ number of iterations

$$\mathbb{E}N = \frac{1}{p_a}.$$

# Rejection technique

## Rejection adaptation

$K$ objects

$$h \geqslant \max_k p_k$$

Generate uniformly on the surface $K \times h$
Accept if the point is under the distribution

## Rejection algorithm

**repeat**
  k= alea(K)
**until** Random . h $\leqslant p_k$
**return** k

alea(K) generate uniformly a number in $\{1, \cdots, K\}$

## Complexity

Acceptance probability $p_a = \frac{1}{hK}$
$N$ number of iterations $\mathbb{E}N = \frac{1}{p_a} = hK$.
Minimal complexity for $h^* = \max_k p_k$.
Uniform distribution $\Rightarrow$ no rejection
Interest : distribution near the uniform distribution

# Rejection technique

## Rejection adaptation

$K$ objects

$$h \geqslant \max_k p_k$$

Generate uniformly on the surface $K \times h$
Accept if the point is under the distribution

## Rejection algorithm

**repeat**
 k= alea(K)
**until** Random . h $\leqslant p_k$
**return** k

alea(K) generate uniformly a number in $\{1, \cdots, K\}$

## Complexity

Acceptance probability $p_a = \frac{1}{hK}$
$N$ number of iterations $\mathbb{E}N = \frac{1}{p_a} = hK$.
Minimal complexity for $h^* = \max_k p_k$.
Uniform distribution $\Rightarrow$ no rejection
Interest : distribution near the uniform distribution

# Rejection technique

## Rejection adaptation

$K$ objects

$$h \geqslant \max_k p_k$$

Generate uniformly on the surface $K \times h$
Accept if the point is under the distribution

## Rejection algorithm

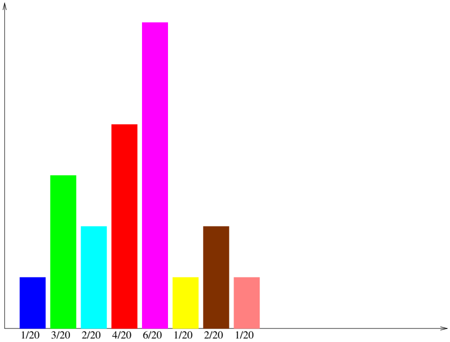
**repeat**
  k= alea(K)
**until** Random . h $\leqslant p_k$
**return** k

alea(K) generate uniformly a number in $\{1, \cdots, K\}$

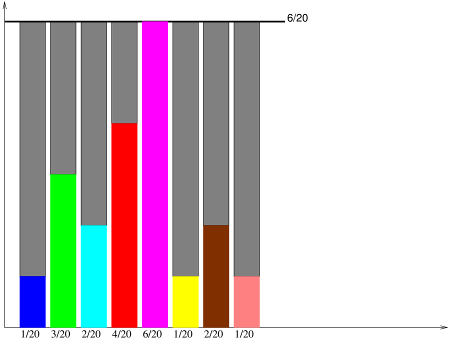
## Complexity

Acceptance probability $p_a = \frac{1}{hK}$
$N$ number of iterations $\mathbb{E}N = \frac{1}{p_a} = hK$.
Minimal complexity for $h^* = \max_k p_k$.
Uniform distribution $\Rightarrow$ no rejection
Interest : distribution near the uniform distribution

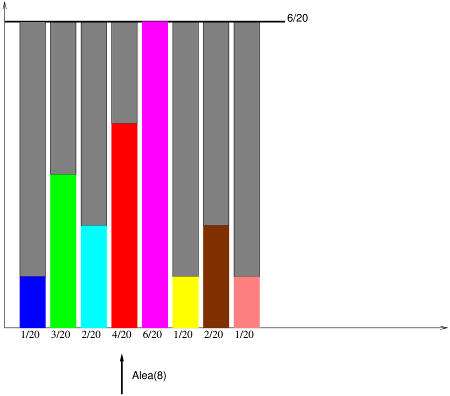
# Rejection Method Applied to Histogram



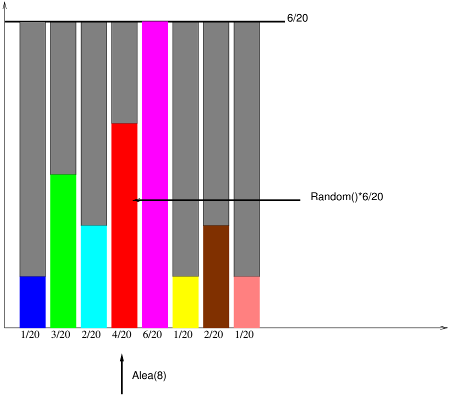
# Rejection Method Applied to Histogram



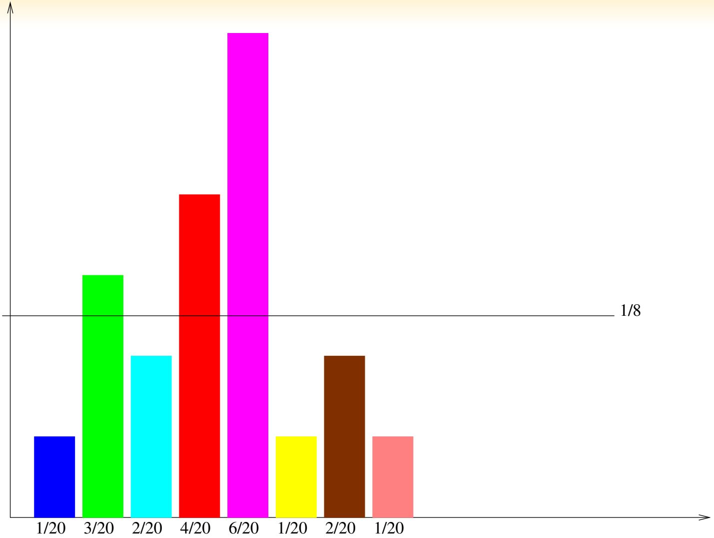
# Rejection Method Applied to Histogram



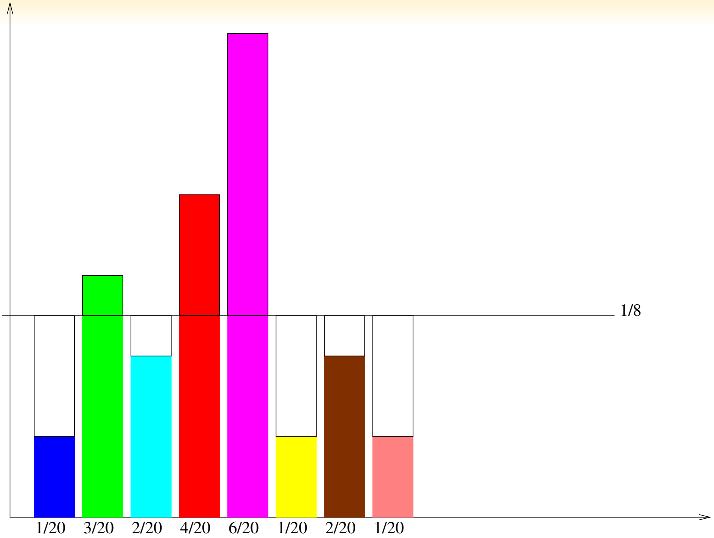
# Rejection Method Applied to Histogram



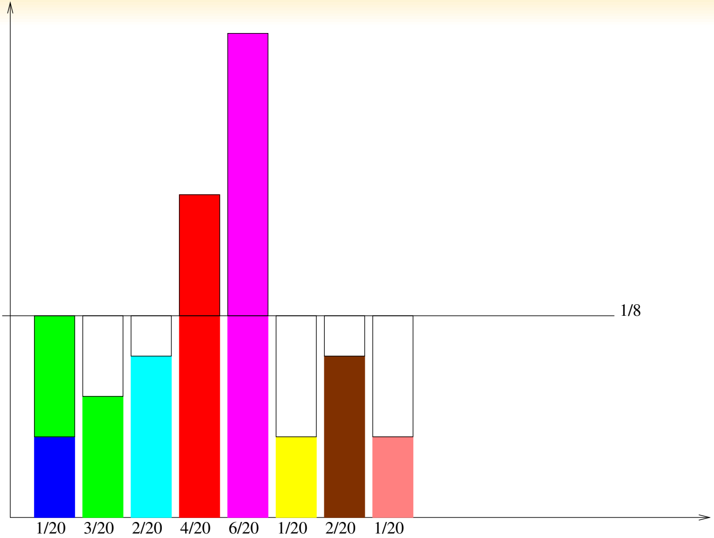
# Aliasing Method



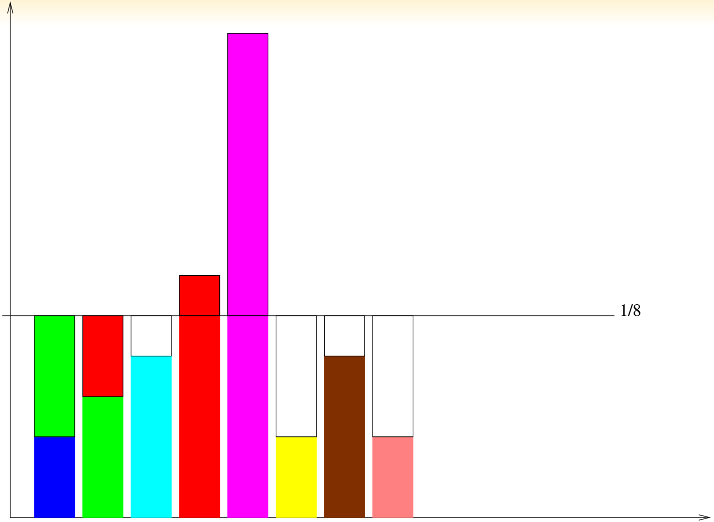
# Aliasing Method



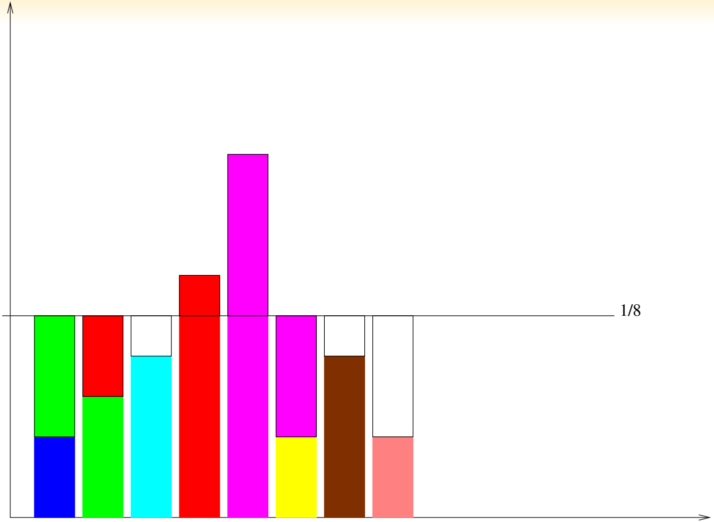
# Aliasing Method



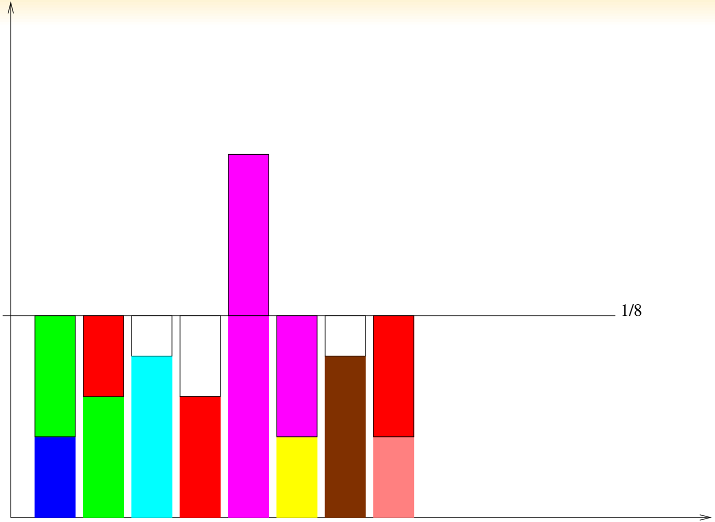
# Aliasing Method



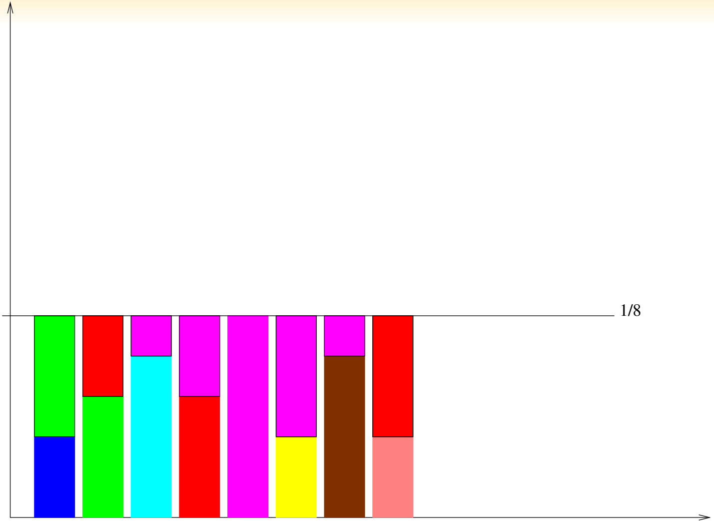
# Aliasing Method



# Aliasing Method



# Aliasing Method



# Aliasing technique

Combine uniform and alias value when rejection

**Initialization**

$K$ objects

list L=$\emptyset$,U=$\emptyset$;
**for** k=1; k$\leqslant$ K; k++ **do**
  P[k]=$p_k$
  **if** P[k] $\geqslant \frac{1}{K}$ **then**
    U=U+$\{$k$\}$;
  **else**
    L=L+$\{$k$\}$;
  **end if**
**end for**

**Alias and threshold tables**

**while** L $\neq \emptyset$ **do**
  Extract $k \in L$
  Extract $i \in U$
  S[k]=P[k]
  A[k]=i
  P[i] = P[i] - ($\frac{1}{K}$-P[k])
  **if** P[i] $\geqslant \frac{1}{K}$ **then**
    U=U+$\{$i$\}$;
  **else**
    L=L+$\{$i$\}$;
  **end if**
**end while**

# Aliasing technique

Combine uniform and alias value when rejection

## Initialization

$K$ objects

list L=$\emptyset$,U=$\emptyset$;
**for** k=1; k$\leqslant$ K; k++ **do**
  P[k]=$p_k$
  **if** P[k] $\geqslant \frac{1}{K}$ **then**
    U=U+$\{$k$\}$;
  **else**
    L=L+$\{$k$\}$;
  **end if**
**end for**

## Alias and threshold tables

**while** L $\neq \emptyset$ **do**
  Extract $k \in L$
  Extract $i \in U$
  S[k]=P[k]
  A[k]=i
  P[i] = P[i] - ($\frac{1}{K}$-P[k])
  **if** P[i] $\geqslant \frac{1}{K}$ **then**
    U=U+$\{$i$\}$;
  **else**
    L=L+$\{$i$\}$;
  **end if**
**end while**

# Aliasing technique : generation

## Generation

k=alea(K)
**if** Random . $\frac{1}{K} \leqslant$ S[k] **then**
  **return** k
**else**
  **return** A[k]
**end if**

## Complexity

Computation time :
- $\mathcal{O}(K)$ for pre-computation
- $\mathcal{O}(1)$ for generation
Memory :
- threshold $\mathcal{O}(K)$ (real numbers as probability)
- alias $\mathcal{O}(K)$ (integers indexes in a tables)

# Aliasing technique : generation

## Generation

k=alea(K)
**if** Random . $\frac{1}{K} \leqslant$ S[k] **then**
  **return** k
**else**
  **return** A[k]
**end if**

## Complexity

Computation time :
- $\mathcal{O}(K)$ for pre-computation
- $\mathcal{O}(1)$ for generation

Memory :
- threshold $\mathcal{O}(K)$ (real numbers as probability)
- alias $\mathcal{O}(K)$ (integers indexes in a tables)

# Outline

# Perfect Sampling of Complex Markov Chains

## Applications

- Finite queuing networks
- Call centers
- Grid/cluster scheduling
- Kitting systems
- Rare event estimation
- Statistical verification of programs

## Modeling

- Poisson systems [Brémaud 1999]
- Discrete vector state-space $\mathcal{X}$
- Event based models

  $$X_{n+1} = \Phi(X_n, e_{n+1}) \text{ , } e_n \in \mathcal{E}$$

  Stochastic recurrence equation
- Independent events (iid)

Provide **independent** samples of **stationary** states.

## PSI2 : a Perfect Sampler

- Library of **monotone** events
- Simulation kernel
- Efficient simulator : polynomial in the model dimension

# Perfect Sampling of Complex Markov Chains

## Applications

- Finite queuing networks
- Call centers
- Grid/cluster scheduling
- Kitting systems
- Rare event estimation
- Statistical verification of programs

## Modeling

- Poisson systems [Brémaud 1999]
- Discrete vector state-space $\mathcal{X}$
- Event based models

$$X_{n+1} = \Phi(X_n, e_{n+1}) \ , e_n \in \mathcal{E}$$

Stochastic recurrence equation
- Independent events (iid)

Provide **independent** samples of **stationary** states.

## PSI2 : a Perfect Sampler

- Library of **monotone** events
- Simulation kernel
- Efficient simulator : polynomial in the model dimension

# Perfect Sampling of Complex Markov Chains

## Applications

- Finite queuing networks
- Call centers
- Grid/cluster scheduling
- Kitting systems
- Rare event estimation
- Statistical verification of programs

## Modeling

- Poisson systems [Brémaud 1999]
- Discrete vector state-space $\mathcal{X}$
- Event based models

$$X_{n+1} = \Phi(X_n, e_{n+1}) \ , e_n \in \mathcal{E}$$

  Stochastic recurrence equation
- Independent events (iid)

Provide **independent** samples of **stationary** states.

## PSI2 : a Perfect Sampler

- Library of **monotone** events
- Simulation kernel
- Efficient simulator : polynomial in the model dimension

# Perfect Sampling of Complex Markov Chains

## Applications

- Finite queuing networks
- Call centers
- Grid/cluster scheduling
- Kitting systems
- Rare event estimation
- Statistical verification of programs

## Modeling

- Poisson systems [Brémaud 1999]
- Discrete vector state-space $\mathcal{X}$
- Event based models

$$X_{n+1} = \Phi(X_n, e_{n+1})\ ,\ e_n \in \mathcal{E}$$

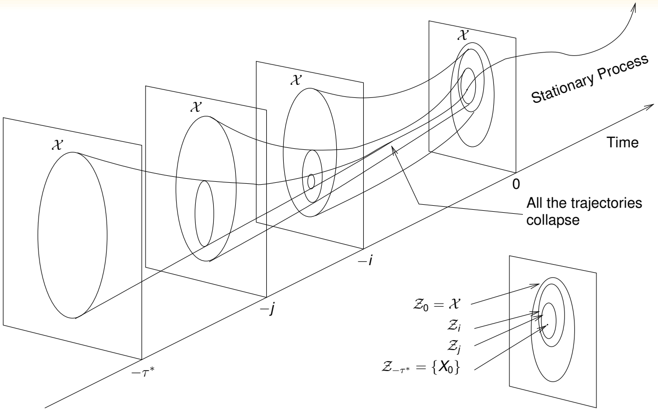
Stochastic recurrence equation
- Independent events (iid)

Provide **independent** samples of **stationary** states.

## PSI2 : a Perfect Sampler

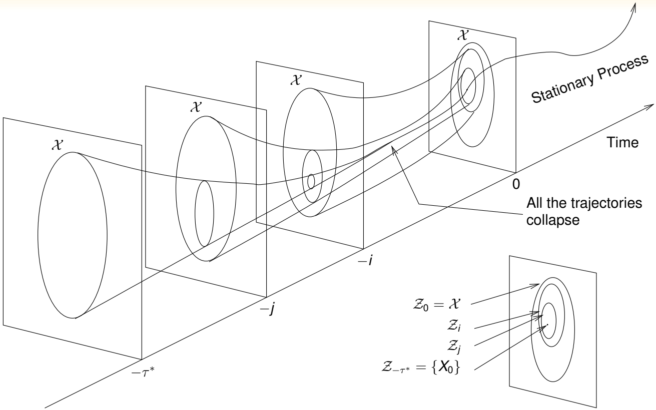
- Library of **monotone** events
- Simulation kernel
- Efficient simulator : polynomial in the model dimension

# Perfect Sampling Principle



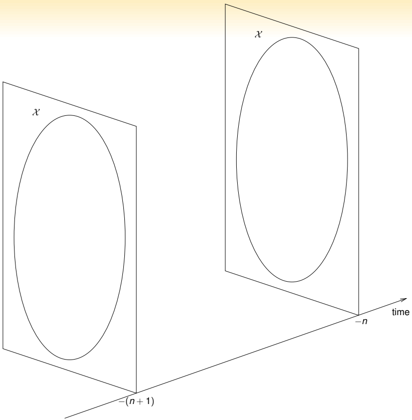

Synchronizing pattern $\Longrightarrow$ finite backward scheme $\tau^* < \infty$

# Perfect Sampling Principle




**Synchronizing pattern $\Longrightarrow$ finite backward scheme $\tau^* < \infty$**

# Monotone Perfect Sampling

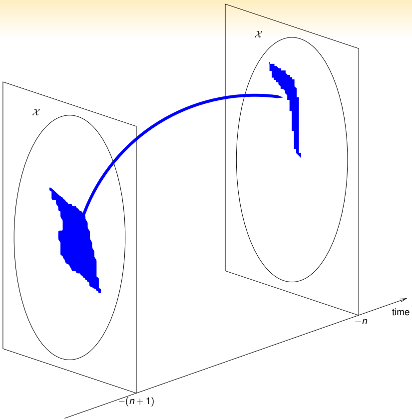



same convergence condition

**complexity in $\mathcal{O}(\mathbb{E}\tau^*)$** $\Rightarrow$ polynomial in model dimension

[QEST 2008]

# Monotone Perfect Sampling

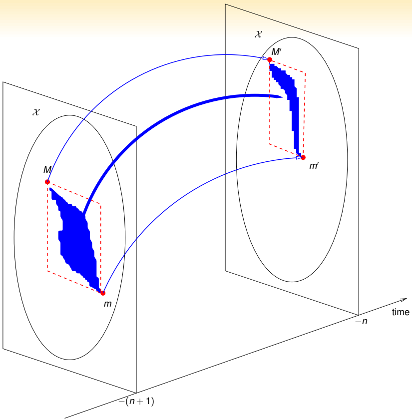



same convergence condition

**complexity in $\mathcal{O}(\mathbb{E}\tau^*)$** $\Rightarrow$ polynomial in model dimension

[QEST 2008]

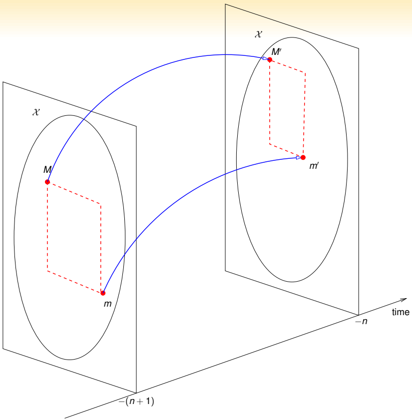
# Monotone Perfect Sampling




same convergence condition
**complexity in** $\mathcal{O}(\mathbb{E}\tau^*)$ $\Rightarrow$ polynomial in model dimension

[QEST 2008]

# Monotone Perfect Sampling




same convergence condition
**complexity in $\mathcal{O}(\mathbb{E}\tau^*)$** $\Rightarrow$ polynomial in model dimension

[QEST 2008]

# Panorama : Markov models

## Finite Monotone Systems

- large class of models : index based routing finite queueing networks
- time complexity : **polynomial** in the dimension of the system

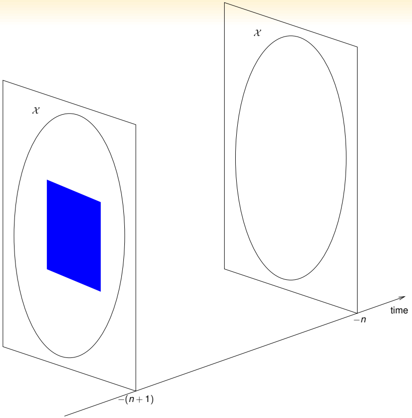
## Finite non-monotone system

- Transition function
  - almost monotone systems : bounding process
  - exhaustive : splitting
  - piecewise linear transitions
- State space extension

## Infinite systems

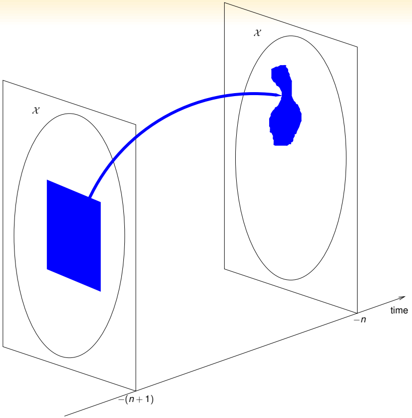
- Monotone transition function
- Non-monotone transitions

# Envelopes Perfect Sampling



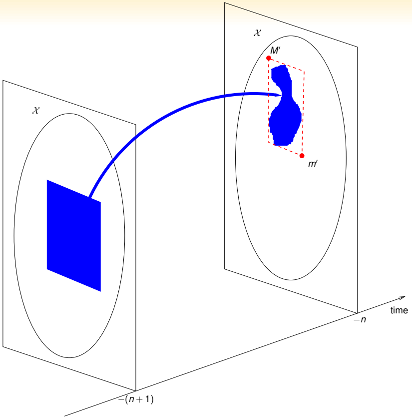

Synchronizing pattern for envelopes
complexity unknown but practically efficient

# Envelopes Perfect Sampling



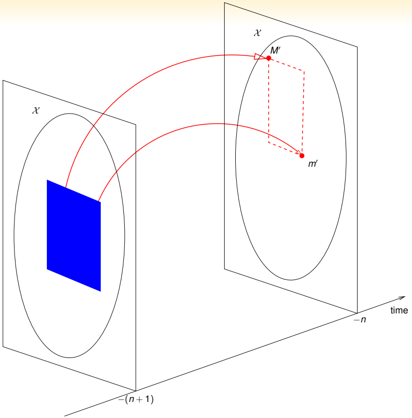

Synchronizing pattern for envelopes
**complexity unknown** but practically efficient

# Envelopes Perfect Sampling



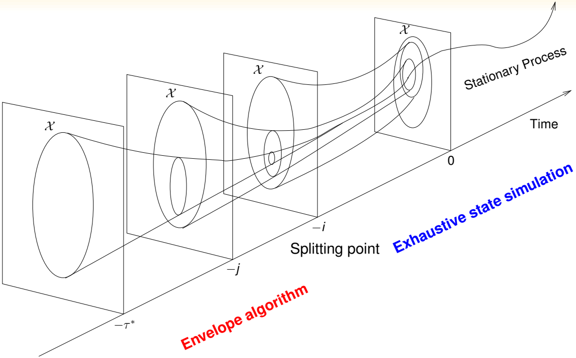

Synchronizing pattern for envelopes
**complexity unknown** but practically efficient

# Envelopes Perfect Sampling




Synchronizing pattern for envelopes
**complexity unknown** but practically efficient

# Envelopes and Splitting Perfect Sampling

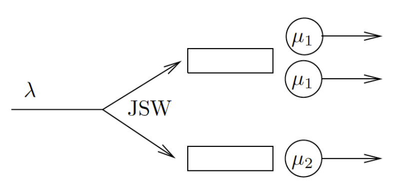



**Guarantees the convergence**
**complexity unknown** but practically more efficient

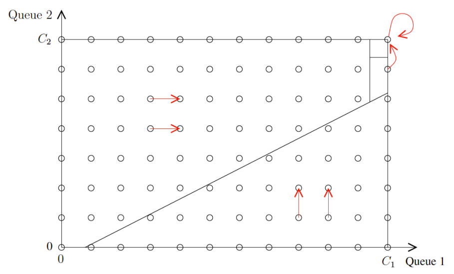
[VALUETOOLS 2008, QEST 2010]

# Computation of Envelopes

## Join the Shortest Weighted Queue




## State space




negative customers, fork and join, batch routing
**general complexity polynomial** (linear programs) but practically $\Rightarrow$ computable less tight bounds

[Performance Evaluation, 2012]

# Outline

# Case Studies