# Performance Modeling of Computer Systems and Networks

*Prof. Vittoria de Nitto Personè*

Discrete-Event Simulation

Università degli studi di Roma Tor Vergata
Department of Civil Engineering and Computer Science Engineering



1

P. 106 discrete

## Multi-Stream Lehmer RNGs

Discrete event simulation (aleatorio)

- Typical DES models have many stochastic components
- Want a unique source of randomness for each component
- One option (poor): multiple RNGs (se ∀ componente ho modello, ha senso pensare che con tante componenti devo usare tante sorgenti di randomness)
- Better option: one RNG with multiple "streams" of random numbers

  one stream per stochastic component

We will partition output from our Lehmer RNG into multiple streams



2

*Case study ssq*

## Arrival and service processes

- two stochastic components: arrival and service
- allocate a different state variable to each

STATO DEL GENERATORE, non del sistema

double GetService(void) (service process original)

```
double GetService(void)
{
    return Uniform(1.0, 2.0);
}
```

```
double GetService(void)
{
    double s;
    static long x = 12345;  (seme partenza)
    PutSeed(x);  (applico)
    s = Uniform(1.0, 2.0);
    GetSeed(&x);  salvo nuovo stato del generatore
    return (s);
}
```

- x represents the current state of the service process





*Case study ssq*

## Arrival and service processes

- Arrival should have its own static variable, initialized differently

```
double GetArrival(void)
{
    static double arrival = START;
    arrival += Exponential(2.0);
    return (arrival);
}
```

```
double GetArrival(void)
{
    static double arrival = START;
    static long x = 54321;
    PutSeed(x);
    arrival += Exponential(2.0);
    GetSeed(&x);
    return (arrival);
}
```

- x represents the current state of arrival process

P 62 slide / 107 discrete



## The Modified Arrival and Service Processes

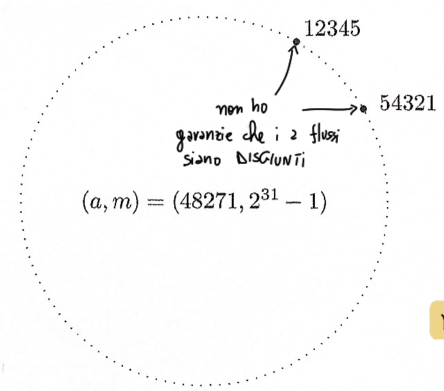
- As modified, arrival and service times are drawn from different streams of random numbers
- Provided the streams don't overlap → the processes are *uncoupled* (disaccoppiati)
- Execution time cost is negligible







- Potential problem: assignment of initial seeds to produce *disjoint* streams
- If states are picked at whim, no guarantee of disjoint streams
- Some initial states may only be a few calls to Random apart!




SOLUZIONE: multistream

## Jump Multipliers

- We will develop a multi-stream version of rng

Theorem
Given $g(x) = ax \bmod m$ (large prime: $m$) and integer j (1<j<m-1)

*jump function*: $g^j(x) = (a^j \bmod m) x \bmod m$

*jump multiplier*: $a^j \bmod m$ (salto di "j" numeri rispetto Lehmer)

If *g(·)* generates $x_0, x_1, x_2, \dots$ then $g^j(\cdot)$ generates $x_0, x_j, x_{2j}, \dots$

- This theorem is the key to creating streams







## Example 1

- If *m* = 31, *a* = 3 and *j* = 6, the jump multiplier is

$$a^j \bmod m = 3^6 \bmod 31 = 16 \quad \text{(salto di 16)}$$

- If $x_0 = 1$, then $g(x) = 3x \bmod 31$ generates:

1, 3, 9, 27, 19, 26, 16, 17, 20, 29, 25, 13, 8, 24, 10, 30, 28, 22, 4, 12, 5, 15, 14, 11, 2, 6, 18, 23, 7, 21, 1, . . .

full period, ed è modulo compatibile (NO OVERFLOW)

- The jump function $g^6(x) = 16x \bmod 31$ generates:

1, 16, 8, 4, 2, 1, . . . disgiunti
$x_0$ $x_6$ $x_{12}$ . . .

## Example 1

- If $m$ = 31, $a$ = 3 and $j$ = 6, the jump multiplier is

$$a^j \ mod \ m = 3^6 \ mod \ 31 = 16$$

- If $x_0$ = 1, then $g(x) = 3x \ mod \ 31$ generates:

1, 3, 9, 27, 19, 26, 16, 17, 20, 29, 25, 13, 8, 24, 10, 30, 28,
22, 4, 12, 5, 15, 14, 11, 2, 6, 18, 23, 7, 21, 1, . . .

- The jump function $g^6(x) = 16x \ mod \ 31$ generates:

1, 16, 8, 4, 2, 1, . . .

- I.e., the first sequence is $x_0$, $x_1$, $x_2$,…; the second is $x_0$, $x_6$, $x_{12}$,…







## Example 1

$m$ = 31, $a$ = 3, $x_0$ = 1

1, 3, 9, 27, 19, 26, 16, 17, 20, 29, 25, 13, 8, 24, 10, 30, 28,
22, 4, 12, 5, 15, 14, 11, 2, 6, 18, 23, 7, 21, 1, . . .

$x_0$ = 1, 3, 9, 27, 19, 26,
$x_6$ = 16, 17, 20, 29, 25, 13,
$x_{12}$ =8, 24, 10, 30, 28, 22,
$x_{18}$ =4, 12, 5, 15, 14, 11,
$x_{24}$ =2, 6, 18, 23, 7, 21,

- The jump function $g^6(x) = 16x \ mod \ 31$ generates:

1, 16, 8, 4, 2, 1, . . .

## Using the jump function

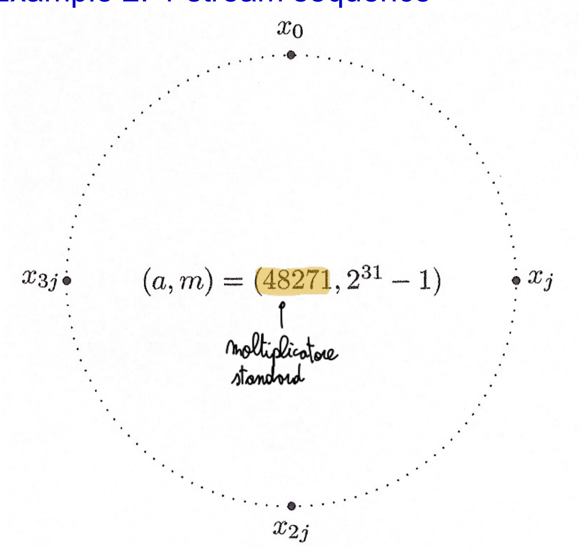
- First, compute the jump multiplier $a^j \; mod \; m$ (one time cost)
- Then, $g^j(\cdot)$ allows jumping from $x_0$ to $x_j$ to $x_{2j}$ to …
- The user supplies ONE initial seed
- If *j* is chosen well, $g^j(\cdot)$ can "plant" additional initial seeds
- Each planted seed corresponds to a different stream
- Each planted seed is *separated* by j calls to `Random`







## Example 2: 4-stream sequence



## An appropriate jump multiplier

- Consider $256 = 2^8$ different streams of random numbers (voglio 256 flussi)
- Partition the RNG output sequence into 256 disjoint subsequences of equal length
- Find the largest $j < 2^{31}/2^8 = 2^{23}$ such that the jump multiplier is modulus-compatible, $j$ lo voglio più grande possibile: $48271^j \bmod m$ sia modulo compatibile
- $g^j(x) = (48271^j \; mod \; m) x \; mod \; m$ can be implemented via algorithm 1 (2.2.1 in the book)
- Then $g^j(x)$ can be used to plant the other 255 initial seeds
- Possibility of stream overlap is minimized (though not eliminated!)

**Algorithm 1**

```
t = a * (x % q) - r * (x / q);          /* t = γ(x) */
if (t > 0)
        return (t);                     /* δ(x) = 0 */
else
        return (t + m);                 /*  δ(x) = 1 */
```







## Maximal Modulus-Compatible Jump Multipliers

- *Maximal jump multiplier*: maximize the distance between streams, $a^j$ mod $m$ where $j$ is the largest integer less than $\lfloor m/s \rfloor$, $s$ number of streams, such that $a^j$ mod $m$ is modulus compatible

Example 2 (cont.) $m = 2^{31} - 1$, $a = 16807$

| # of streams $s$ | $\lfloor m/s \rfloor$ | jump size $j$ | jump multiplier $a^j$ mod $m$ |
|---|---|---|---|
| 1024 | 2097151 | 2082675 | 97070 |
| 512 | 4194303 | 4170283 | 44857 |
| 256 | 8388607 | 8367782 | 22925 |
| 128 | 16777215 | 16775552 | 40509 |

## Library rngs

- Upward-compatible multi-stream replacement for rng
- Provides 256 streams, indexed 0 to 255 (0 is the default)
- Only one stream is active at any time
- 6 available functions:
  - `Random(void)`: to use the standard Lehmer generator
  - `PutSeed(long x)`: to set the state of the active stream
  - `GetSeed(long *x)`: to obtain the state of the active stream
  - `TestRandom(void)`: to test the implementation correctness
  - `SelectStream(int s)`: to define the active stream (lo scelgo)
  - `PlantSeeds(long x)`: “plants” one seed per stream