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

Quanto ci mette un ordine ad arrivare, viene modellato con probabilità.

Multi-Stream Lehmer RNGs

- Typical DES models have many stochastic components
- Want a unique source of randomness for each component
- One (poor) option: multiple RNGs
- 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

Il generatore è sempre di Lehmer, con più "streams". Non posso usare lo stesso flusso per arrivi e servizi, anche perchè creerei correlazione tra due processi che sono slegati. Devo usare un flusso separato per ogni componente (un pezzo della ruota per arrivi, un altro per servizi). Devo cercare di capire come usare 'tale ruota' per diverse componenti stocastiche che devono rimanere indipendenti. E' alla base del multi-stream.

Arrival and service processes

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

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

```
double GetService(void)
{
    double s;
    static long x = 12345;
    PutSeed(x);
    s = Uniform(1.0, 2.0);
    GetSeed(&x);
    return (s);
}
```

solo la prima volta!

- x represents the current state of the service process

lo stato è l'ultimo valore generato, in pratica quando richiamo la funzione questa riparte dall'ultimo valore estratto e 'procede'.

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;  cambio seme
    PutSeed(x);
    arrival += Exponential(2.0);
    GetSeed(&x);
    return (arrival);
}
```

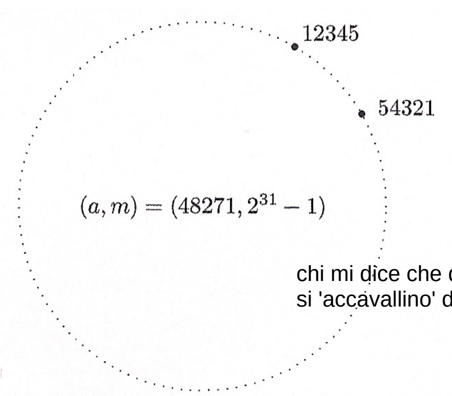
solo la prima volta!

- x represents the current state of arrival process

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*
- 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!



chi mi dice che questi due punti non
si 'accavallino' da un certo punto in poi?

Jump Multipliers

- We will develop a multi-stream version of rng

Theorem

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

jump function: $g^j(x) = \left(a^j \; mod \; m\right) x \; mod \; m$

jump multiplier: $a^j \; mod \; m$

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

mi sposto su sequenze di lunghezza 'j'.

- This theorem is the key to creating streams

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:

$\underbrace{1, 16, 8, 4, 2,}$ 1, . . .

genera sequenza di lunghezza j-1 = 5

e' come avere sei generatori disgiunti, anche se in realtà ne sto usando uno!
Questa tecnica è migliore di usare 6 generatori di Lehmer.

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 $x_0, x_1, x_2, \ldots$; the second is $x_0, x_6, x_{12}, \ldots$

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,

Devono però bastarmi questi numeri!

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

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

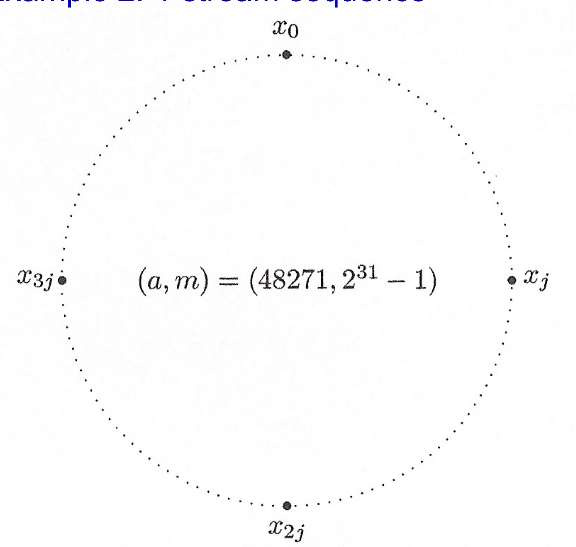
Using the jump function

- 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 uso un seed e poi applico la funzione salto.
- 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`

ogni seed dista 'j' dall'altro.

voglio dividere la ruota in 4 flussi separati, con generatore che 'salti' da un flusso all'altro, cioè tra i vari x(i), che sono pezzi disgiunti.

Example 2: 4-stream sequence



An appropriate jump multiplier

- Consider 256 = 2^8 different streams of random numbers
- 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
- $g^j(x) = \left(48271^j \bmod m\right) x \bmod m$ 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, $a^j \bmod m$ where j is the largest integer less than $\lfloor m/s \rfloor$, s number of streams, such that $a^j \bmod m$ is modulus compatible

Example 2 (cont.)

	# of streams s	lunghezza sottosequenza $\lfloor m/s \rfloor$	intero più grande ma minore di m/s jump size j	jump multiplier $a^j \bmod m$
	1024	2097151	2082675	97070
	512	4194303	4170283	44857
standard	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 da dove 'partire'.
  - `PlantSeeds(long x)`: “plants” one seed per stream

a partire dal primo seme pianta tutti gli altri