Performance Modeling of Computer Systems and Networks

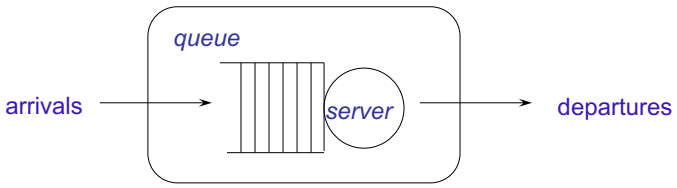
Prof. Vittoria de Nitto Personè

Discrete-Event Simulation
examples

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

1

Single Server Queue



Arrival times: a_i

15 47 71 111 123 152 166 226 310 320

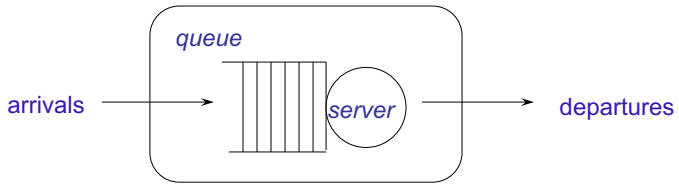
Pseudo-random generators

Service times: s_i

43 36 34 30 38 40 31 29 36 30

2

Single Server Queue



- assume **service times** are between 1.0 and 2.0 minutes
  - The distribution within this range is unknown valori tra 1 e 2 equiprobabili
  - Without further knowledge, we assume no time is more likely than any other

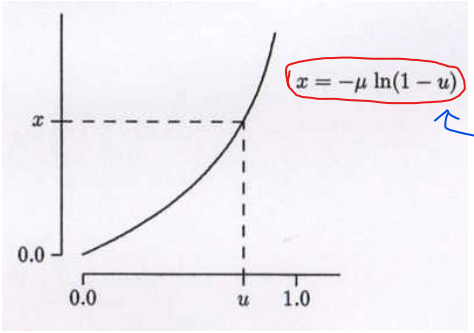
→ Uniform(1.0, 2.0)

3

Exponential distribution

- In general, it is unreasonable to assume that all possible values are equally likely
- Frequently, small values are more likely than large values
- We need a non-linear transformation that maps $0.0 \rightarrow 1.0$ to $0.0 \rightarrow \infty$

this is the most frequently used function
$\mu > 0$ is a parameter that "control" the frequency of large values in respect of the small ones
La funzione "parte" da mu.
Se mu cresce, la media (1/mu) decresce



"u" è generato da Lehmer.

4

• the transformation is monotone increasing, one-to-one

$0 < u < 1 \Leftrightarrow 0 < (1 - u) < 1$
$\Leftrightarrow -\infty < \ln(1 - u) < 0$
$\Leftrightarrow 0 < -\mu \ln(1 - u) < \infty$
$\Leftrightarrow 0 < x < \infty$

in questo snippet di codice mu è la media.

```
double Exponential(double μ)        /* use μ > 0.0 */
{
  return (-μ * log(1.0 - Random()));
}
```

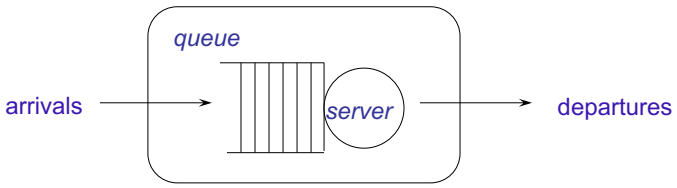
genero Lehmer

• the parameter μ specifies the sample mean

5

Fino ad ora, dovevamo avere le tracce (quindi tempi scritti), adesso li generiamo noi.

Single Server Queue



Arrival times: a_i

• use the exponential function for the interarrival times

$a_i = a_{i-1} + \textit{Exponential}(\mu)$; i = 1, 2, 3, . . . , n

ultimo istante di arrivo

prima c'era "r(i)"
cioè l'interarrivo.

Service times: s_i generato come uniforme (1.0, 2.0)

Uniform(1.0, 2.0)

6

- program ssq2 is an extension of ssq1
  - arrival times are drawn from *Exponential*(2.0)
  - service times are drawn from *Uniform*(1.0, 2.0)

rispetto a prima, adesso non usiamo più la traccia dei file, bensì distribuzioni di probabilità (quindi li genero).

7

trace-driven simulation

```
#include <stdio.h>

#define FILENAME   "ssq1.dat"     /* input data file */
#define START      0.0

  double GetArrival(FILE *fp)   /* read an arrival time */
{   double a;
    fscanf(fp, "%lf", &a);
    return (a);}

double GetService(FILE *fp)   /* read a service time */
{   double s;
    fscanf(fp, "%lf\n", &s);
    return (s);}
```

8

ssq2.c

distribution-driven simulation

```
#include <stdio.h>
#include <math.h>
#include "rng.h"
#define LAST          10000L    /* number of jobs processed */
#define START         0.0

double Exponential(double m)                    /* ------*
{return (-m * log(1.0 - Random())); }             m > 0.0
                                                 ------ */
double Uniform(double a, double b)              /* ------*
{return (a + (b - a) * Random());    }              a < b
                                                * ------*/

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

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

incremento arrival con tempo di inter-arrivo di media 2

uniforme tra 1 e 2

- the program generates all first-order statistics

$\bar{r}, \bar{w}, \bar{d}, \bar{s}, \bar{l}, \bar{q}, \bar{x}$

interarrivo medio,
tempo risposta medio,
attesa media in coda,
servizio medio

utilizzazione
numero nella coda,
numero totale nel sistema,

trace-driven simulation = distribution-driven simulation

```
int main(void)
{  FILE   *fp;                  /* input data file */
   long   index     = 0;       /* job index */
   double arrival   = START;   /* arrival time*/
   double delay;               /* delay in queue*/
   double service;              /* service time*/
   double wait;                 /* delay + service*/
       double departure = START;   /* departure time*/
   struct {                    /* sum of ...  */
       double delay;     /*delay times */
       double wait;      /*wait times*/
       double service;   /*service times */
       double interarrival; /*   interarrival times */
   } sum = {0.0, 0.0, 0.0};
```


11

trace-driven simulation

```
fp = fopen(FILENAME, "r");
if (fp == NULL) {
   fprintf(stderr,"Cannot open input file %s\n",FILENAME);
   return (1);  }
while (!feof(fp)) {
```

distribution-driven simulation

```
PutSeed(123456789);
while (index < LAST) {
```

inizializzo generatore

LAST: ultimo job che voglio simulare

12

```
                    trace-driven: fp
while (index < LAST) {
   index++;
   arrival      = GetArrival();
   if (arrival < departure)
       delay  = departure - arrival; /* delay in queue  */
   else  delay      = 0.0;           /* no delay  */
   service = GetService();
   wait  = delay + service;
   departure    = arrival + wait;   /* time of departure */
   sum.delay   += delay;
   sum.wait    += wait;
   sum.service += service;
}
sum.interarrival = arrival - START;}
```


13

```
printf("\nfor %ld jobs\n", index);
printf("   average interarrival time = %6.2f\n", sum.interarrival / index);
printf("   average wait ..... = %6.2f\n", sum.wait / index);
printf("   average delay ..... = %6.2f\n", sum.delay / index);
printf("   average service time .... = %6.2f\n", sum.service / index);
printf("   average # in the node ... = %6.2f\n", sum.wait / departure);
printf("   average # in the queue .. = %6.2f\n", sum.delay / departure);
printf("   utilization ..... = %6.2f\n", sum.service / departure);
return (0);}
```


14

Siamo in un caso a coda infinita, servente singolo, λ = 0,5 j/s exp, μ = 0.67, vogliamo trovare gli indici. Come?

Da KP: $E[T_q] = \frac{\rho}{1-\rho}\left(\frac{c^2+1}{2}\right) \cdot E[S]$ attenzione, perchè media e varianza si riferiscono al servente, che ha tempi UNIFORMI.

Questo vuol dire che: E[S] = $\frac{a+b}{2}$, $\sigma^2[x] = \frac{(b-a)^2}{12}$ NB: posso usare KP solo se ρ<1, altrimenti l'equazione sarebbe negativa.

$\rho = \lambda E[S] = 0,75$ $c^2 = \frac{\sigma^2}{E[S]^2} = \frac{1}{27} = 0.\overline{037}$ molto più piccolo dell'equivalente esponenziale con stessa media.

$E[T_q] = 2,\overline{3}$

$E[T_s] = E[T_q] + E[S] = 2,\overline{3} + 1,5 = 3,8\overline{3}\ s$

Per le popolazioni medie, basta usare Little ricordando che N = λ.T calcolo E[Nq] in quanto conosco E[Tq] e poi aggiungo l'utilizzazione.

Example 1

- The "theoretical" averages using *Exponential*(2.0) (rate 0.5 j/s) arrivals and *Uniform*(1.0, 2.0) (rate 0.67) service times are

$\bar{r}$	$\bar{w}$	$\bar{d}$	$\bar{s}$	$\bar{l}$	$\bar{q}$	$\bar{x}$
2.00	3.83	2.33	1.50	1.92	1.17	0.75

exact analytical results, No simulation!

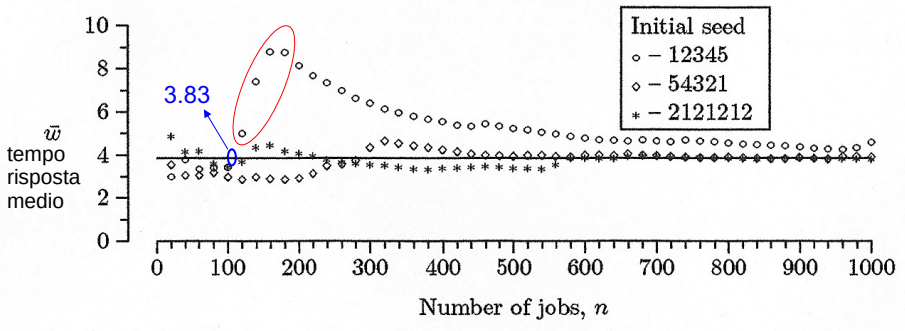
$\rho = 0,75$

- Although the server is busy 75% of the time, on average there are approximately 2 jobs in the service node $E[N_s] = 1.9167$
- A job can expect to spend more time in the queue than in service infatti E[S] = 1.5 mentre E[Tq] = 2.3
- To achieve these averages, many jobs must pass through node ovvero, per ottenere questi risultati, il numero di job nel centro deve essere molto alto! Infatti, avendo usato KP, queste sono medie asintotiche.

server occupato 75%, cioè 1/4 del tempo vuoto.
Il sistema parte da vuoto.

- The accumulated average wait was printed every 20 job

rho<1, si stabilizza, ma quanto ci vuole? in questo caso servono parecchi job, infatti all'inizio il seme influenza di molto il grafico.



- The convergence of *w* is slow, erratic, and dependent on the initial seed

ogni "punto" è la media calcolata fino a quel determinato punto.

bisogna sempre specificare quale comportamento stiamo studiando!

quando diventa stazionario

- the program can be used to study the *steady-state* behavior
  - Will the statistics converge independent of the initial seed?
  - How many jobs does it take to achieve steady-state behavior?

ovvero la fase precedente alla stazionarietà (la fase iniziale), come fatto prima, con varie simulazioni per vederne il comportamento.

- the program can be used to study the *transient* behavior
  - Fix the number of jobs processed and replicate the program with the initial state fixed
  - Each replication uses a different initial rng seed

ovviamente tutte le simulazioni devono avere LO STESSO STATO INIZIALE
(es: sistema iniziale vuoto), e SEME DIVERSO (sennò starei copiando le simulazioni).

Steady-state analysis studio STAZIONARIO

a partire da un certo seme, come variano le statistiche all'aumentare del seme.

	$\bar{r}$	$\bar{w}$	$\bar{d}$	$\bar{s}$	$\bar{l}$	$\bar{q}$	$\bar{x}$	
theoretical	2.00	3.83	2.33	1.50	1.92	1.17	0.75	
n=10	2.85	1.74	0.39	1.35	0.59	0.13	0.45	
n=100	2.06	3.16	1.67	1.48	1.50	0.80	0.71	
n=1000	2.03	3.44	1.94	1.50	1.69	0.96	0.74	seed=123456789
n=10000	2.02	3.86	2.36	1.50	1.91	1.17	0.74	
n=100000	2.00	3.85	2.35	1.50	1.92	1.17	0.75	
n=1000000	2.00	3.81	2.31	1.50	1.90	1.15	0.75	
n=1000000	2.00	3.84	2.34	1.50	1.92	1.17	0.75	seed=1

per un milione di jobs il seme diverso sembra portare poca variabilità.
Gli indici scaturiti dai generatori risultano essere quelli meno variabili

n=10	2.13	2.36	0.75	1.62	1.02	0.32	0.69	seed=1
n=10	1.48	2.37	0.87	1.50	1.24	0.46	0.79	seed=987654321
n=10	1.49	1.89	0.49	1.40	1.12	0.29	0.83	seed=2121212121

Transient analysis

Nell'analisi transiente (a inizio simulazione) la scelta del seed ha un maggior impatto.

Nel progetto vengono richieste entrambe.

Simulating an initial steady state

Parto da uno stato stabile, e non vuoto.

$\bar{d} = 2.33$ delay, tempo attesa in cosa, preso dalla KP.

`departure=3` governa stato sistema, se = 0 vuol dire che il primo job che arriva trova vuoto.
se Departure = 3 simulo uno stato stabile, perchè?

$a_1=a_0+expo(2)=0+0.8=0.8$ primo interarrivo, cioè primo job arriva a 0.8
primo completamento a 3.

$0.8 < 3$ c'è attesa di 2.2 s

$d_1=3-0.8=2.2$ attesa in coda, simula abbastanza bene lo stato stabile.

$s_1=Uniform(1,2)=1.3$ tempo servizio

$w_1=2.2+1.3=3.5$ tempo di risposta: attesa in coda + tempo servizio

$c_1=0.8+3.5=4.3$ quando job1 esce dal centro

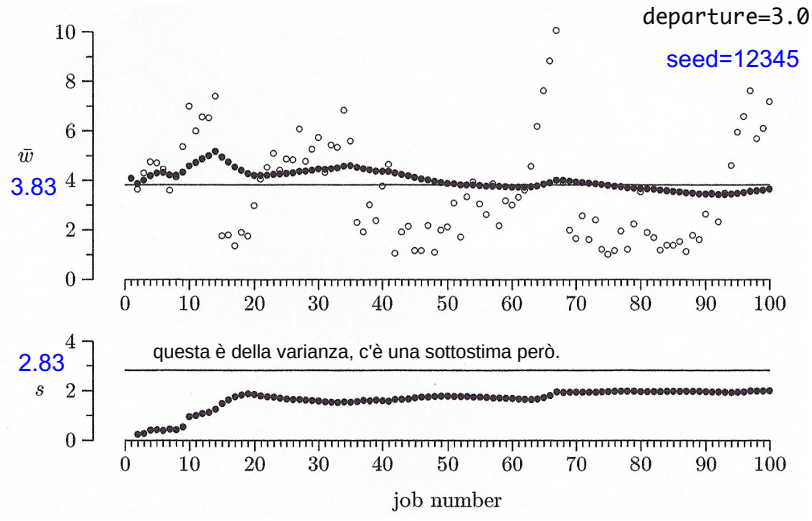
19

simulando questo stato, NON VUOTO, ecco come cambia la curva:

● media ○ singola osservazione del job

DE simulation
Sample statistics

Ho simulato lo stato STAZIONARIO sin dall'inizio.



20

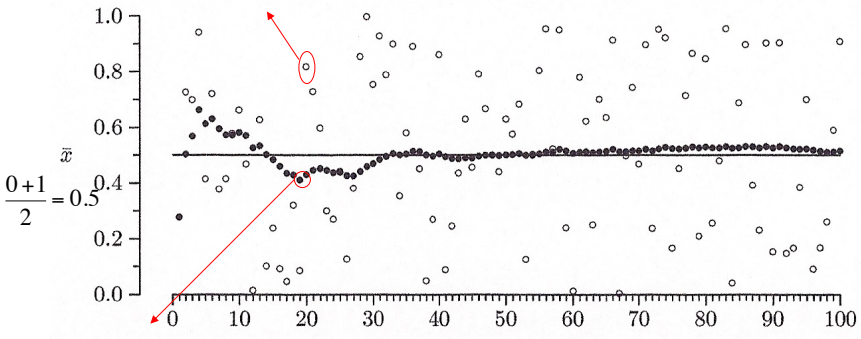
Se ho tempo di risposta lungo, anche il job che viene dopo di me lo sperimenterà.
Ovvero se vediamo il primo grafico ho alcuni "gruppi" sopra la media, e altri gruppi sotto la media.
Questo è il problema della statistica campionaria rispetto la teorica. Cioè statisticamente NON E' INDIPENDENTE, anzi c'è una correlazione.

Il primo grafico introduce un BIAS sul secondo grafico, che dovremo correggere.

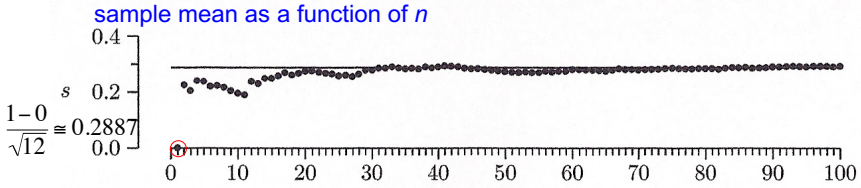
Generazione di 100 variabili uniformi tra 0 e 1, usando generatore. Sono sparse, conferma la bontà del generatore.
Anche nella varianza (secondo grafico) non c'è BIAS, e presto tende al valore teorico, perchè non c'è correlazione tra i pallini bianchi.

Uniform(0, 1)

sample values



sample mean as a function of *n*



s=0 if *n*=1

21

Serial correlation

- *Independence*: each x_i value does not depend on any other point
- Time-sequenced DES output is typically not independent(serie storiche)
- E.g.: wait times of consecutive jobs have positive *serial correlation*
- Example: Consider output from ssq2
  - *Exponential*(2) interarrivals, *Uniform*(1,2) service
- wait times $w_1, w_2, \ldots, w_{100}$, have high positive serial correlation
  - The correlation produces a bias in the standard deviation

22

p.101 discrete

Example 2

- assume that jobs arrive at random with a steady-state arrival rate of 0.5 jobs per minute
- assume that Job service times are "composite" with two components:
  - the *number* of service tasks is 1 + *Geometric*(0.9) media geometrica: 1/0.9
  - the *time* (in minutes) per task is *Uniform*(0.1, 0.2) media: $\frac{0.1 + 0.2}{2} = 0.15$

```
double GetService(void)
{
long k;
double sum = 0.0;
long tasks = 1 + Geometric(0.9);
for (k = 0; k < tasks; k++)
   sum += Uniform(0.1, 0.2);
return (sum);
}
```


23

invece di avere un job che chiede un tempo uniforme, ho job composti da almeno un task dove ogni task ha tempo 0.15 di media. In media 10 task, che corrisponde a E[S] =10*0.15 = 1.5 (mi ricollego all'esercizio delle slide prima). Ciò che cambia è la variabilità!

- The theoretical steady-state statistics for this model are

$\bar{r}$	$\bar{w}$	$\bar{d}$	$\bar{s}$	$\bar{l}$	$\bar{q}$	$\bar{x}$
2.00	5.77	4.27	1.50	2.89	2.14	0.75
	(3.83	2.33		1.92	1.17)	

exact analytical results, No simulation!

- The arrival rate, service rate, and utilization are identical to the previous case (example 1)
- The other four statistics are significantly larger
- performance measures are sensitive to the choice of service time distribution

ciò che è cambiata è la distribuzione dei servizi.

24

A simple inventory system

- Distributes items from current inventory to customers
- Customer demand is discrete
- Simple: one type of item
  - Inventory review is periodic
  - Items are ordered, if necessary, only at review times
  - (s, S) are the min,max inventory levels, 0≤s<S

Fisso

variabile, per vedere come cambia il comportamento

Simply Inventory System

- Program sis2 has randomly generated demands using an *Equilikely*(a, b) random variate

non realistica, uso la geometrica, anche se pure lei non è perfetta.

- Using random data, we can study transient and steady-state behaviors

sis1.c

```
#include <stdio.h>

#define FILENAME  "sis1.dat"
#define MINIMUM   20
#define MAXIMUM   80
#define STOP     100
#define sqr(x)    ((x) * (x))

long GetDemand(FILE *fp)
{
   long d;
   fscanf(fp, "%ld¥n", &d);
   return (d);}
```

sis2.c

unico input: domanda di articoli in una settimana. in sis1 da traccia,
qui è una Equilikely equivalente discreta della uniform.

```
#include <stdio.h>
#include "rng.h"

#define MINIMUM   20
#define MAXIMUM   80
#define STOP     100      /* 100 weeks = about 2 years*/
#define sqr(x)    ((x) * (x))


long Equilikely(long a, long b)
{  return (a + (long) ((b - a + 1) * Random()));}

long GetDemand(void)
{
        return (Equilikely(10, 50)); }
```

i :	1	2	3	4	5	6	7	8	9	10	11	12
d_i :	30	15	25	15	45	30	25	15	20	35	20	30

27

```
int main(void)
{
   long index     = 0;
   long inventory = MAXIMUM;
   long demand;
   long order;
   struct {
        double setup;
        double holding;  /*inventory hold (+) */
        double shortage; /*inventory short (-) */
        double order;
        double demand;
   } sum = { 0.0, 0.0, 0.0, 0.0, 0.0 };

    PutSeed(123456789);
```

28

```
while (index < STOP) {
  index++;
  if (inventory < MINIMUM) {
       order = MAXIMUM - inventory;
       sum.setup++;
       sum.order    += order;
  }
  else order         = 0;
  inventory += order;     /* there is no delivery lag */    demand  =
  GetDemand();
  sum.demand     += demand;
  if (inventory > demand)
       sum.holding  += (inventory - 0.5 * demand);
  else {
       sum.holding  += sqr(inventory) / (2.0 * demand);
       sum.shortage += sqr(demand - inventory) / (2.0 * demand);
  }
  inventory      -= demand;
}
```


29

```
if (inventory < MAXIMUM) {
       order = MAXIMUM - inventory;
       sum.setup++;
       sum.order+= order;
       inventory      += order;
  }

...
```


30

for 100 time intervals with an average demand of 27.68
and policy parameters (s, S) = (20, 80)

average order ..... = 27.68
setup frequency ..... = 0.36
average holding level .... = 44.81
average shortage level ... = 0.14

- trace file sis1.dat contains data for n=100 time intervals
- with (s, S)=(20, 80)

$\bar{o} = \bar{d} = 29.29 \qquad \bar{u} = 0.39 \qquad \bar{l}^{+} = 42.40 \qquad \bar{l}^{-} = 0.25$

31

```
int main(void)
{  long seed;
   long index     = 0;
   long inventory = MAXIMUM;
   long demand;
   long order;
   struct {
        double setup;
        double holding;  /*inventory held (+) */
        double shortage; /*   inventory short (-)   */
        double order;
        double demand;
   } sum = { 0.0, 0.0, 0.0, 0.0, 0.0 };

PutSeed(-1);  uso clock del sistema, per la ripetitività mi serve però conoscerlo.
GetSeed(&seed); lo prelevo con getSeed
printf("\nwith an initial seed of %ld", seed);
```


32

for 100 time intervals with an average demand of 27.68
and policy parameters (s, S) = (20, 80)

average order ..... = 27.68
setup frequency ..... = 0.36
average holding level .... = 44.81
average shortage level ... = 0.14

with an initial seed of 1333437895 estratto dal clock
for 100 time intervals with an average demand of 31.00
and policy parameters (s, S) = (20, 80)

average order ..... = 31.00
setup frequency ..... = 0.40
average holding level .... = 43.39
average shortage level ... = 0.37

i risultati cambiano, perchè è come se stessimo vedendo scenari diversi.

Simply Inventory System

media: $\frac{a + b}{2} = 30$

if (a, b) = (10, 50) and (s, S) = (20, 80), then the approximate steady-state satistics are

$\bar{d}$	$\bar{o}$	$\bar{u}$	$\bar{l}^+$	$\bar{l}^-$
30.00	30.00	0.39	42.86	0.26

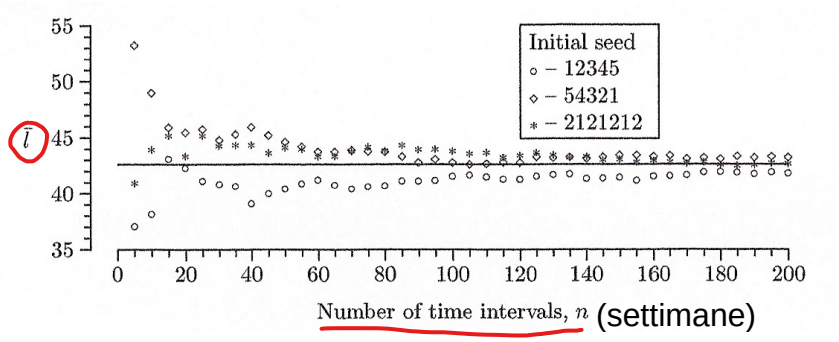
valore teorico prodotto da sis2.

(trace-driven prodotto con l'uso di una traccia, quindi dati presi da file.

$\bar{o} = \bar{d} = 29.29$ $\bar{u} = 0.39$ $\bar{l}^+ = 42.40$ $\bar{l}^- = 0.25$)

tendenza alla stabilità

The average inventory level $\bar{l} = \bar{l}^+ - \bar{l}^-$
approaches steady state after several hundred time intervals



Convergence is slow, erratic, and dependent on the initial seed

35

Se uso lo stesso seme, cambiando solo 's' (che è ciò che studiamo, ovvero livello minimo scorte). Non avrebbe senso cambiare 's' & 'seme'.

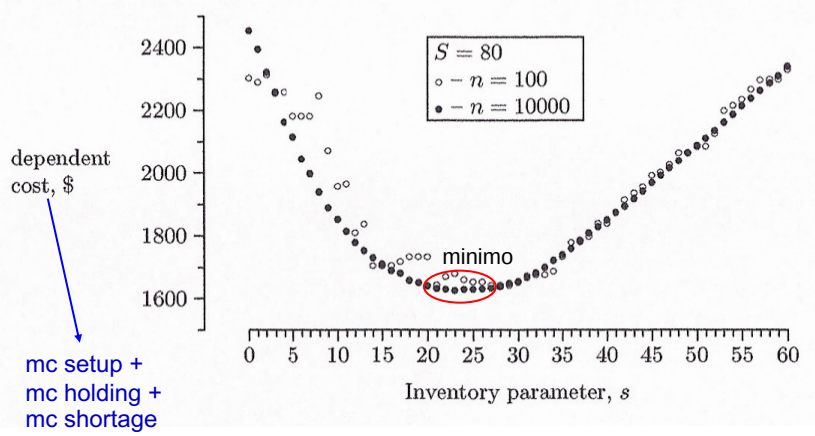
- using a fixed initial seed guarantees the exact same demand sequence (in the example 12345)
  - any changes to the system are caused solely by the change of s
  - a steady state study of this system is unreasonable:
    - all parameters would have to remain fixed for many years
    - when n=100, we simulate approximately 2 years
    - when n=10000, we simulate approximately 192 years

Se dovessi analizzare la variabilità, dovrei farlo a parità di traccia, non avrebbe senso vederla in uno scenario diverso.

36

Nell'esempio, l'unità di misura sono le settimane. Con n = 100, simuliamo due anni, ma chi mi dice che in due anni i costi non cambino? per n = 100000 sono quasi due secoli.

- if we fix S, we can find the optimal cost by varying s
- recall that the dependent cost ignores the fixed cost of each item



mc setup +
mc holding +
mc shortage

37

Se 's' = 23, ovviamente dobbiamo valutare se questo 's' sia accettabile dalle condizioni di contorno.

Statistical Considerations
(sect. 3.1.3)

- *variance reduction:*

  (intuitive approach) use the same random numbers

  (We kept the same initial seed and changed only s)

NOTE:

transient behavior will always have some inherent uncertainty

- *Robust Estimation*:

  when a data point is not sensitive to small changes in assumptions

  - values of s close to 23 have essentially the same cost
  - Would the cost be more sensitive to changes in S or other assumed values?

38

Exercises

- derive analytical results on p.12
- study program ssq2.c; run it and compare output with the results on p.12
- Exercises: 3.1.1, 3.1.2, 3.1.4, 3.1.5, 3.1.6