Discrete-Event Simulation

Lawrence M. Leemis and Stephen K. Park, Discrete-Event Simul A First Course, Prentice Hall, 2006

Hui Chen

Computer Science
Virginia State University
Petersburg, Virginia

February 15, 2017

Introduction

- Programs ssq1 and sis1 are *trace-driven* discrete-event simulations
  - Both rely on input data from an external source
- These realizations of naturally occurring stochastic processes are limited
- Cannot perform "what if" studies without modifying the data
- Solution
  - Convert the single server service node and the simple inventory system to utilize *randomly generated input*
  - Use a random-number generator to produce the randomly generated input
  - Discrete-event simulation programs using the randomly generated input does not depend on external trace data

Single Queue Service Node: Revisited

- Need two stochastic assumptions
  - arrival times
  - service times
- The assumptions governs how arrival and service times are randomly generated in discrete-event simulation programs

Example: Generating Service Times: Uniform Distribution

- Service time
  - Range: between 1.0 and 2.0
  - Distribution within the range?
    Without further knowledge, we assume no time is more likely than any other
  - To generate service times: use $u = \mathit{Uniform}(1.0, 2.0)$ random variate

Example: Generating Service Times: Uniform Distribution

Is it reasonable to assume that service times are uniformly distributed, e.g., service times are generated using $u = \mathit{Uniform}(1.0, 2.0)$ random variate?

Example: Generating Service Times: Uniform Distribution

Is it reasonable to assume that service times are uniformly distributed, e.g., service times are generated using $u = \mathit{Uniform}(1.0, 2.0)$ random variate?

It depends.

Example: Generating Service Times: Uniform Distribution

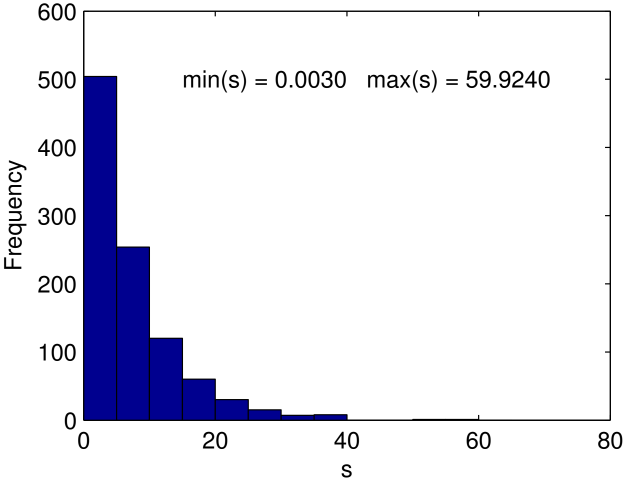
Is it reasonable to assume that service times are uniformly distributed, e.g., service times are generated using $u = \mathit{Uniform}(1.0, 2.0)$ random variate?

It depends.

In most applications, it is unrealistic to assume service times are uniformly distributed.

Service Time in *ssq1.dat* Trace Data

Is service times in *ssq1.dat* uniformly distributed?



Example: Generating Service Times: 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
- Need a non-linear transformation that maps $0 \to 1$ to $0 \to \infty$ since $0 < u = Uniform(0, 1) < 1$

Example: Generating Service Times: Exponential Distribution

- A common nonlinear transformation is

$$x = -\mu ln(1-u) \tag{1}$$

- The transformation is monotone increasing, one-to-one, and onto

$$\begin{aligned}
0 < \mu < 1 &\iff 0 > -u > -1 && (2)\\
&\iff 0+1 > -u+1 < -1+1 && (3)\\
&\iff 1 > 1-u > 0 && (4)\\
&\iff ln(1) > ln(1-u) > ln(0) && (5)\\
&\iff 0 > ln(1-u) > \infty && (6)\\
&\iff 0 < -ln(1-u) < \infty && (7)\\
&\iff 0 < -\mu ln(1-u) < \infty && (8)\\
&\iff 0 < x < \infty && (9)
\end{aligned}$$

Example: Generating Service Times: Exponential Distribution

- The common nonlinear transformation $x = -\mu ln(1-u)$ is monotone increasing, one-to-one, and onto

$$0 < \mu < 1 \Longleftrightarrow 0 < -\mu ln(1-u) < \infty \Longleftrightarrow 0 < x < \infty \qquad (10)$$

which generates *Exponential*(μ) random variate

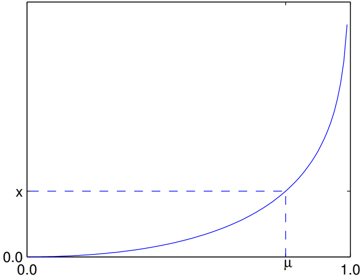


Figure: Exponential-variate-generation Geometry

Example: Generating Service Times: Exponential Distribution

- The common nonlinear transformation

$$x = -\mu ln(1-u) \tag{11}$$

generates *Exponential*(μ) random variate

- Note that $0 < u < 1$ and

$$\begin{aligned}
& \int_0^1 -\mu ln(1-u)du = -\mu \int_0^1 ln(1-u)du && (12)\\
= & -\mu \int_0^1 -ln(1-u)d(1-u) = \mu \int_0^1 ln(1-u)d(1-u) && (13)\\
= & \mu\{ln(1-u)(1-u)|_0^1 - \int_0^1 (1-u)dln(1-u)\} && (14)\\
= & \mu\{0-(1-u)\frac{1}{1-u}(1-u)|_0^1\} && (15)\\
= & -\mu(1-u)\frac{1}{1-u}(1-u)|_0^1 = -\mu(1-u)|_0^1 && (16)\\
= & \mu && (17)
\end{aligned}$$

i.e., the parameter μ specifies the sample mean

Generating *Exponential*(μ) Random Variate

Definition 3.1.1 ANSI C Function for *Exponential*(μ)

```
double Exponential(double μ)
{
  return - μ * log(1.0 - Random());
}
```

where *Random*() generates $u = Uniform(0, 1)$ random variate and μ is the sample mean.

Example: Generating Service Times: Exponential Distribution

In the single-server service node simulation, we use $\mathit{Exponential}(\mu_s)$ to generate service times,

$$s_i = \mathit{Exponential}(\mu_s); \quad i = 1, 2, 3, \ldots, n \tag{18}$$

where μ_s is the sample mean of service times.

Example: Generating Interarrival Times: Exponential Distribution

In the single-server service node simulation, we use $\mathit{Exponential}(\mu_a)$ to generate interarrival times,

$$a_i = a_{i-1} + \mathit{Exponential}(\mu_a); \quad i = 1, 2, 3, \ldots, n \tag{19}$$

where μ_a is the sample mean of interarrival times.

Example: Recap

- Inter-arrival times
  - Generating $u = Uniform(a, b)$ random variate
  - Generating $u = Exponential(a)$ random variate
- Service times
  - Generating $u = Uniform(a, b)$ random variate
  - Generating $u = Exponential(a)$ random variate

Simulation Program *ssq2*

- Program *ssq2* is an extension of *ssq1*
  - Interarrival times are drawn from $Exponential(2.0)$
  - Service times are drawn from $Uniform(1.0, 2.0)$
- The program generates job-averaged and time-averaged statistics
  - $\bar{r}$: average interarrival time
  - $\bar{w}$: average wait
  - $\bar{d}$: average delay
  - $\bar{s}$: average service time
  - $\bar{l}$: average # in the node
  - $\bar{q}$: average # in the queue
  - $\bar{x}$: server utilization

Exercise L4-1

In this exercise, you are required to complete the following tasks,

- Develop *ssq2* by revising *ssq1* program.
- Compile and run the *ssq2* program.
- When writing the program, meet the following,
  - Interarrival times are drawn from $Uniform(0.0, 6.0)$
  - Service times are drawn from $Exponential(2.0)$
- Submission: the source code of *ssq2*, the results of the program, and evidence that your program appears to be correct.

Example 3.1.3: Theoretical Result from Analytic Model

- The theoretical averages for a single-server service node using *Exponential*(2.0) inter-arrivals and *Uniform*(1.0, 2.0) service times are (Gross and Harris, 1985),

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

- Although the server is busy only 75% of the time, on average there are approximately two jobs in the service node
- A job can expect to spend more time in the queue than in service
- To achieve these averages, many jobs must pass through node

Example 3.1.3: Results from Simulation Program *ssq2*

- The accumulated average wait was printed every 20 jobs

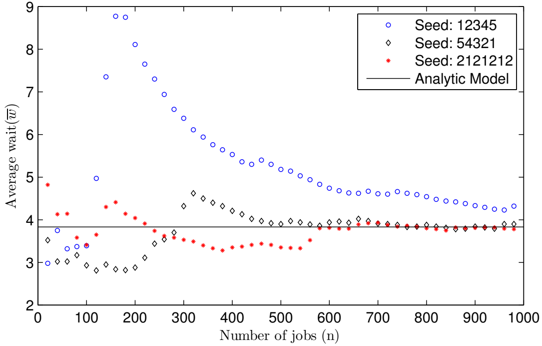


Figure: Average wait times

- The convergence of $\overline{w}$ is slow, erratic, and dependent on the initial seed

Use of Program *ssq2*

- 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?
- It 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

Exericse L4-2

You are required to reproduce the figure in slide 20. You may take steps below (using the C/C++ program as an example),

- Convert the main function *int main(void)* to function *void SimulateOnce(long seed, long last)*.
  - *seed*: seed of RNG
  - *last*: the number of jobs to process
- Add the *main* function in which you call *SimulateOnce* with *seed* and *last* in a loop with *last* as the loop variable to simulate with the number of jobs as 20, 40, . . . , 1000.
- Format the output in the "CSV" format.
- Run the program and graph the results.
- Submission: program source code, running results, and graph.

Geometric Random Variables

- The *Geometric*(p) random variate is the discrete analog to a continuous *Exponential*(μ) random variate
  Let $x = Exponential(\mu) = \mu ln(1-\mu)$, $y = \lfloor x \rfloor$, and $p = Pr(y \neq 0)$

$$y = \lfloor x \rfloor \neq 0 \iff x \geq 1 \tag{20}$$
$$\iff \mu ln(1-\mu) \geq 1 \tag{21}$$
$$\iff ln(1-\mu) \leq -1/\mu \tag{22}$$
$$\iff 1-\mu \leq e^{-1/\mu} \tag{23}$$

Since $1-\mu$ is also $Uniform(0.0, 1.0)$ and $p = Pr(y \neq 0) = e^{-1/\mu}$
Finally, since $\mu = -1/ln(p)$, $y = \lfloor ln(1-\mu)/ln(p) \rfloor$

Generating *Geometric*(p) Random Variates

Definition 3.1.2 ANSI C Function for *Geometric*(p)

```
long Geometric(double p)   use 0.0 < p < 1.0
{
  return (long)(log(1.0 - Random()) / log(p));
}
```

- *Random*() generates $u = Uniform(0, 1)$ random variate.
- The mean of a *Geometric*(p) random variate is $p/(1 - p)$
  - If p is close to zero then the mean will be close to zero
  - If p is close to one, then the mean will be large

Example 3.1.4: Composite Service Model

Now consider a composite service model

- 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 $1 + Geometric(0.9)$
  - The time (in minutes) per task is $Uniform(0.1, 0.2)$

Example 3.1.4: Composite Service Model

ANSI C Function for the Composite Service Model

```
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);
}
```

Example 3.1.4: Composite Service Model: Analytic Model

- 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

- The arrival rate, service rate, and utilization are identical to Example 3.1.3 (See slide 19)
- The other four statistics are significantly larger
- Performance measures are sensitive to the choice of service time distribution

Simple Inventory System: Example 3.1.5

- Program *sis2* has randomly generated demands using an *Equilikely*(a, b) random variate
- Using random data, we can study transient and steady-state behaviors
- If $(a, b) = (10, 50)$ and $(s, S) = (20, 80)$, then the approximate steady-state statistics are

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

Exercise L4-3

In this exercise, you are required to complete the following tasks,

- Compile and run the *sis2* program. Document the results.
- Make a copy of the *sis2* program, revise it to meet the following,
  - The demand is drawn from *Geometric*(0.967742)

  and then compile and run the program.
- Submit your work including both version of the *sis2* program and the results of both runs in Blackboard

Effects of Number of Time Intervals and Seed of RNG

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

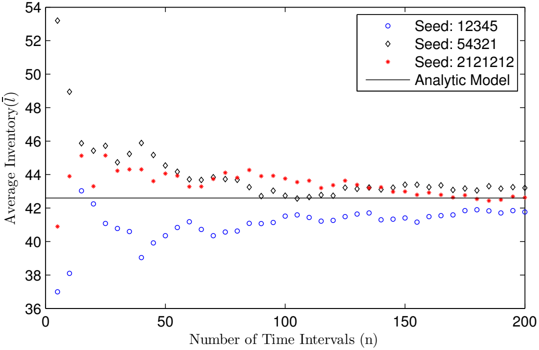


Figure: Number of Time Intervals (n)

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

Exercise L4-4

You are required to reproduce the figure in slide 30. You may take steps below (using the Java program as an example),

- Convert the main function *public static void main(String[] args)* to function *public static void SimulateOnce(long seed, long stop)*.
  - *seed*: seed of RNG; *stop*: the number of intervals to process
  - Format the output in the "CSV" format
- Add the *public static void main(String[] args* function in which you call *SimulateOnce* with *seed* and *stop* in a loop with *stop* as the loop variable to simulate with the number of intervals as 5, 10, 15, . . ., 200.
- Run the program and graph the results
- Submission: program source code, results, and graph.

Example 3.1.7: Optimal Inventory Policy

- If we fix S, we can find the optimal cost by varying s

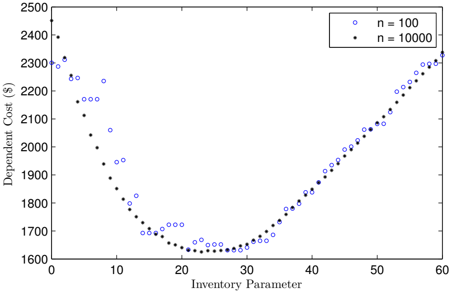


Figure: Dependent Cost for (s, S) Inventory System

where $c_{setup} = \$1,000$, $c_{hold} = 25$, $c_{short} = 700$, $min(DependentCost) = \$1,624.86$, and $s = 24$.

- Recall that the dependent cost ignores the fixed cost of each item

Example 3.1.7: Discussion

- Using a fixed initial seed guarantees the exact same demand sequence
  - 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

Statistical Considerations

- Example 3.1.7 illustrates two consideration
  - Variance reduction
  - Robust estimation
- With Variance Reduction, we eliminate all sources of variance except one
  - Transient behavior will always have some inherent uncertainty
  - We kept the same initial seed and changed only s
- Robust Estimation occurs when a data point that 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?

Exercise L4-5

You are required to reproduce the figure in slide 32.

Hints (using the Java program as an example):

- Revise

  ```
  public static void SimulateOnce(long seed, long stop)
      throws IOException { ......
  ```

  to

  ```
  public static void SimulateOnce(long seed, long stop, int slower)
      throws IOException { ......
  ```

  where *slower* is s is (s, S) in the inventory system.

- In the main method/function, call the *SimulateOnce* method/function with $stop = 100$ and $stop = 10000$, respectively in two loops whose loop variable changes from $slower = 0$ to $slower = 60$ with increment 1.
- Let $c_{setup} = \$1,000$, $c_{hold} = 25$, and $c_{short} = 700$. Compute the dependent cost in an Excel workbook. Graph the cost versus s for the two *stop* values.

$$C_{dependent} = c_{setup}\bar{u} + c_{hold}\bar{l}^{+} + c_{short}\bar{l}^{-}$$

- Submission: both the program and the Excel workbook.

Summary

- Discrete-Event Simulations: random variate vs. trace
- Revisited SSQ
- Revisited SIS
- Variance reduction and robust estimation