Discrete-Event Simulation: Examples

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

Hui Chen

Computer Science
Virginia State University
Petersburg, Virginia

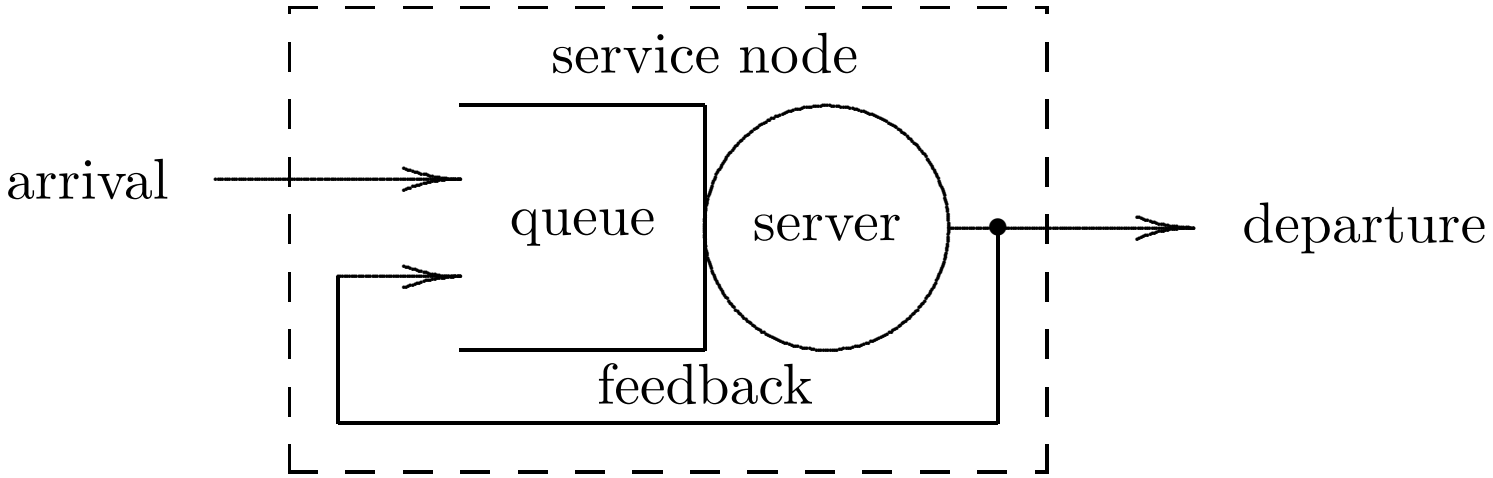
March 22, 2017

Table of Contents

SSQ with Immediate Feedback

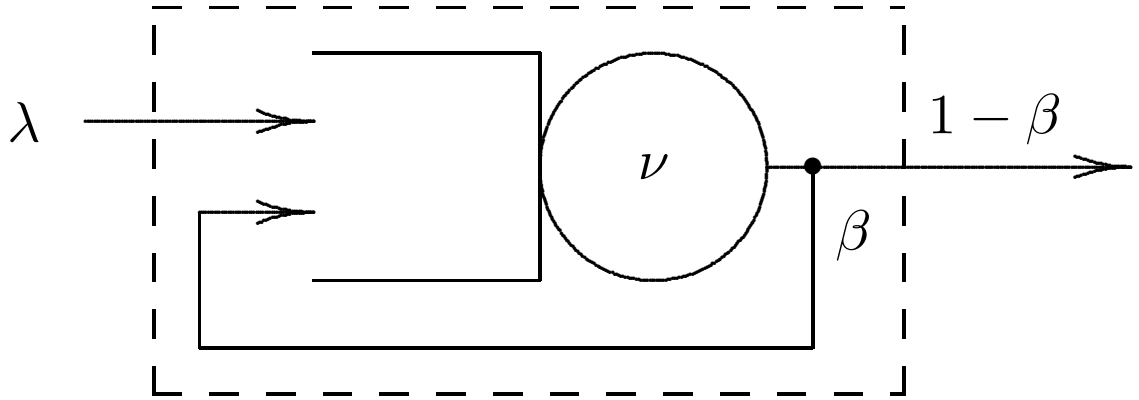
We now consider *immediate feedback* in the single-server service node model, i.e.,

- the possibility that the service a job just received was incomplete or otherwise unsatisfactory and the job feedback immediately to request service once again.
- Completion of service and departure now have different meanings
  - At the completion of service, jobs either depart the service node forever or immediately feed back and once again seek service.



Model Considerations

- As it completes its service, each job departs the service node with probability $1 - \beta$, or feeds back with probability β
  - When feedback occurs the job joins the queue consistent with the queue discipline
  - The decision to depart or feed back is random with feedback probability β



  - λ is the arrival rate
  - ν is the service rate

Model Considerations

- Feedback is independent of past history
- In theory, a job may feed back arbitrarily many times
- Typically, β is close to 0.0

GetFeedback Method

```
int GetFeedback(double beta) /* 0.0 <= beta < 1.0 */
{
    SelectStream(2);
    if (Random() < beta)
        return (1); /* feedback occurs */
    else
        return (0); /* no feedback */
}
```

Statistical Considertations

- Index $i = 1, 2, 3, \ldots$ counts jobs that enter the service node
  - fed-back jobs are not recounted
- Using this indexing, all job-averaged statistics remain valid
- We must update delay times, wait times, and service times for each feed back
- Jobs from outside the system are merged with jobs from the feedback process by the positive additive factor $\lambda \overline{x} \nu$
- Note that $\overline{s}$ increases with feedback but $1/\nu$ is the average service time per request

Example 3.3.1

job index	1	2	3	4	5	·	6	·	7	8	·	9	...
arrival/ feedback	1	3	4	7	10	**13**	14	**15**	19	24	**26**	30	...
service	9	3	2	4	7	5	6	3	4	6	3	7	...
completion	10	**13**	**15**	19	**26**	31	37	**40**	44	50	53	**60**	...

- At the computational level, some algorithm and data structure is necessary

Algorithm and Data Structure Considerations

- Need to insert fed-back jobs into arrival stream
- Need a data structure to hold fed-back jobs and arrivals

Exercise L7-1 (Hints in Next Slide)

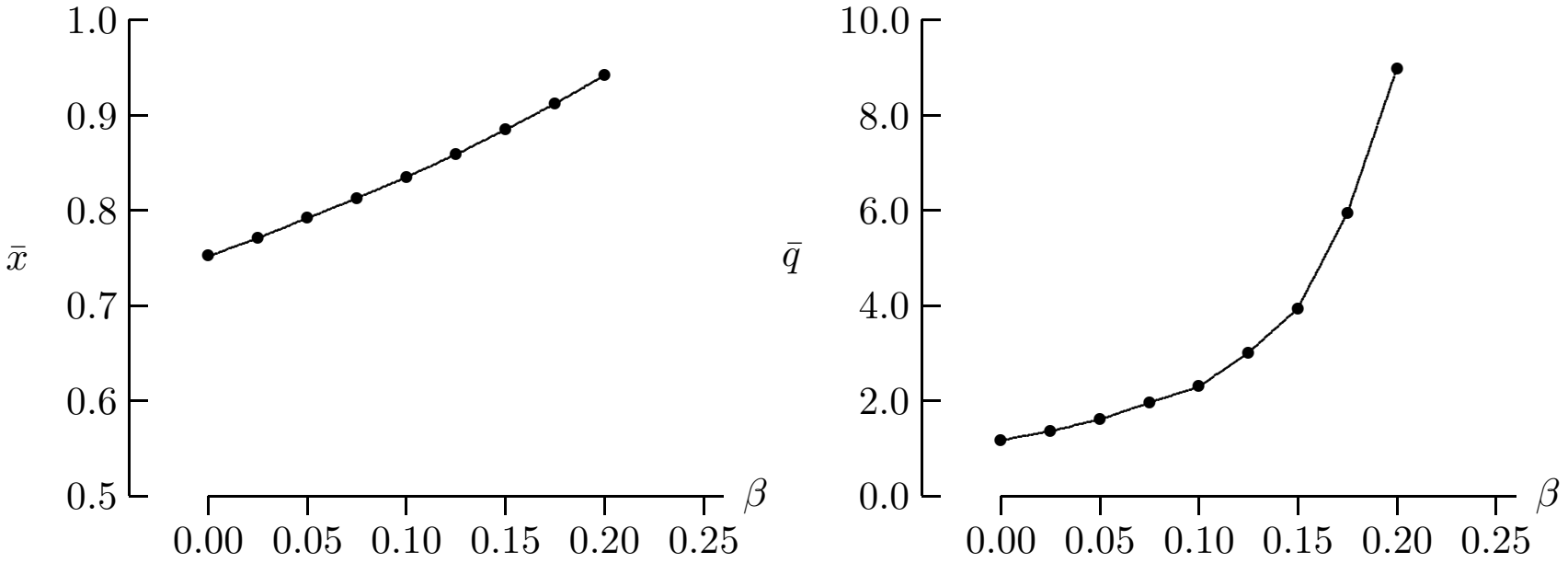
- Make a copy of *ssq2* and name the copy as *ssq2ex332*.
- Modify *ssq2ex332* to account for immediate feedback
  - Use an array to store arrivals including both fed-back and new arrivals
  - hint: how big the array shoud be? which array element contains earliest arrival in the array including the fed-back one?
- Use the following parameters
  - The arrival process has *Exponential*(2.0) random-variate interarrival times
  - The service process has *Uniform*$(1.0, 2.0)$ random-variate service times
  - The feed-back probability is $0.0 \leq \beta < 1.0$. To illustrate the effect of feedback, the modified program is to simulate the operation of a single-server service node with 9 different values of levels of feedback, varied from $\beta = 0.0$ to $\beta = 0.20$
  - In each case, $100,0000$ arrivals are to be simulated.
- Graph utilization $\overline{x}$ as a function of β and the average number in the queue $\overline{q}$ as a function of β

Exercise L7-1: Hints

- Change the main method/function to *SimulateOnce(double feedbackProbability)*
  - Let it print out the output as *CSV* format;
  - Declare an array to hold arrivals including fed-back arrivals
  - Insert arrivals to the array
  - Search earliest arrival from the array
- Add *int GetFeedback(double beta)* method/function in slide 5 to the program
- Add a main method/function that *has* the logic similar to the following,

```
double step = 0.2/8.0, feedbackProbability;  int i;
for (i = 0; i <= 8; i ++) {
    feedbackProbability = step * i;
    SimulateOnce(feedbackProbability);
}
```

Example 3.3.2: Sample Result for Exercise L7-1



- Left-hand side: Utilization $\overline{x}$ versus feedback probability β
- Right-hand side: The average number of jobs in the queue $\overline{q}$ versus feedback probability β

Discussion

- Is *array* a good choice to store arrivals in Exercise L7-1?
- What checks can you do to have an improved confidence on the program and the result?

Flow Balance and Saturation

- Jobs flow into the service node at the average rate of λ
- To remain flow balanced jobs must flow out of the service node at the same average rate
- The average rate at which jobs flow out of the service node is

$$\overline{x}(1-\beta)\nu$$

- Flow balance is achieved when $\lambda = \overline{x}(1-\beta)\nu$
- Saturation occurs when $\overline{x} = 1$ or as $\beta \to 1 - \lambda/\nu = 0.25$

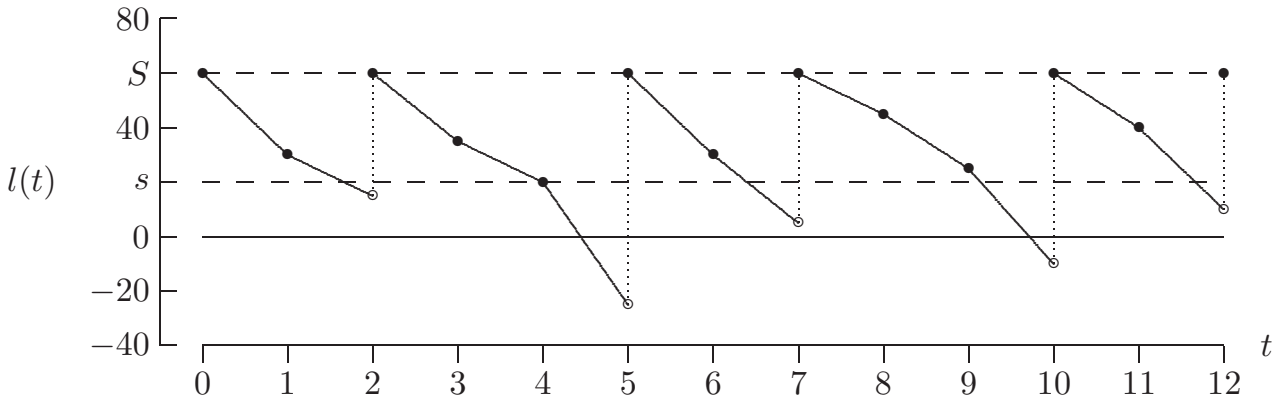
Simple Inventory System with Delivery Log

An extension to the periodic-review simple-inventory-system model

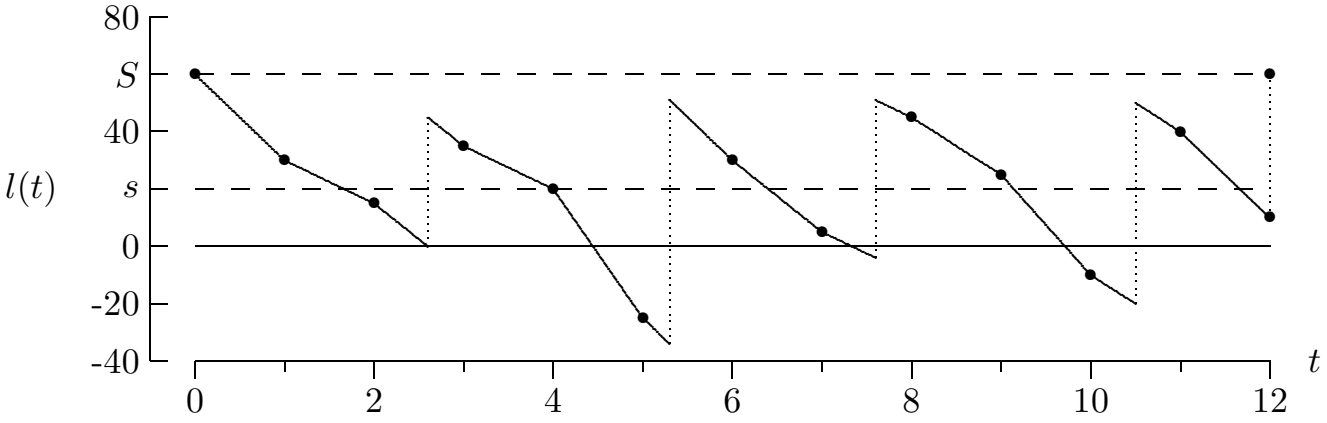
- *Delivery lag* (or *lead time*)
  - An inventory replacement order placed with the supplier will not be delivered immediately
  - There will be a *lag* between the time an order is placed and the time the order is delivered
  - Lag is *assumed* to be random and independent of order size

Inventory Levels With and Without Delivery Lag

- Without lag, inventory jumps occur only at inventory review times

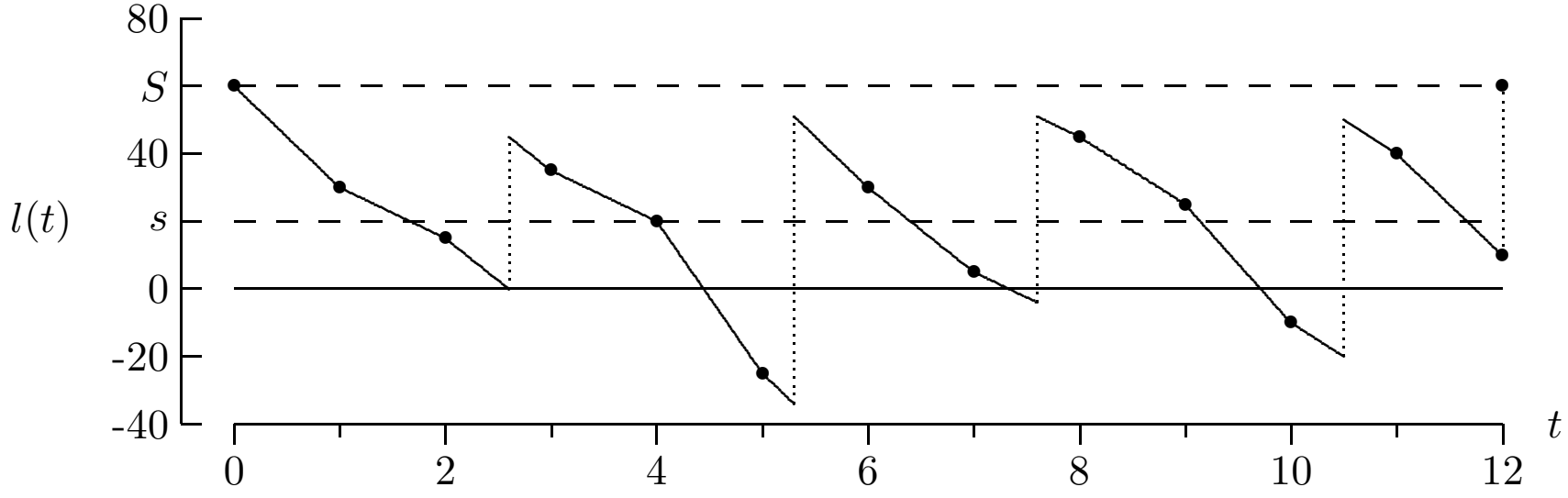


- With delivery lag, inventory jumps occur at arbitrary times



SIS with Lag

- The last order is *assumed* to have no lag
- We *assume* that orders are delivered before the next inventory review
- With these assumptions, there is no change to the specification model. However, there is a significant change in how the system statistics are computed.



SIS with Lag: Statistical Consideration

There is a significant change in how the system statistics are computed.

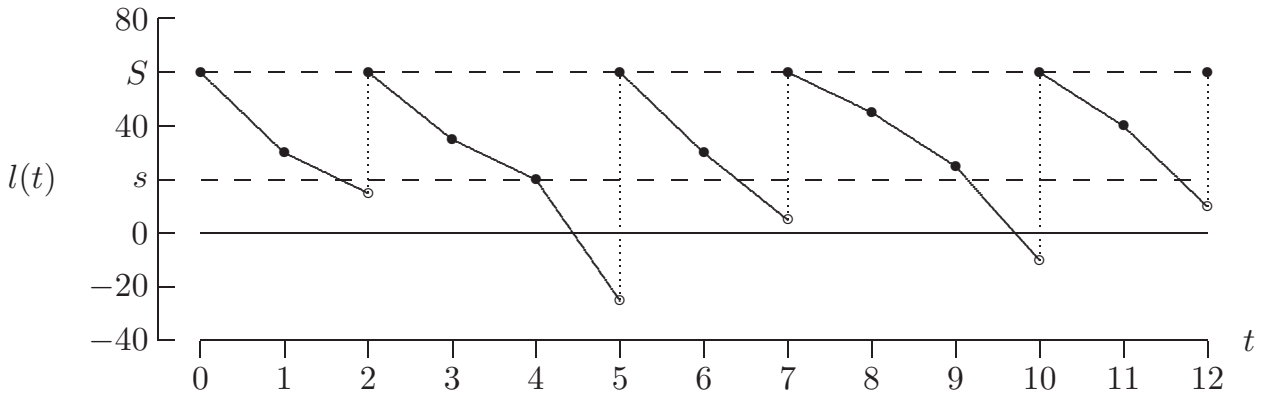


Figure: Inventory level *without* delivery lag

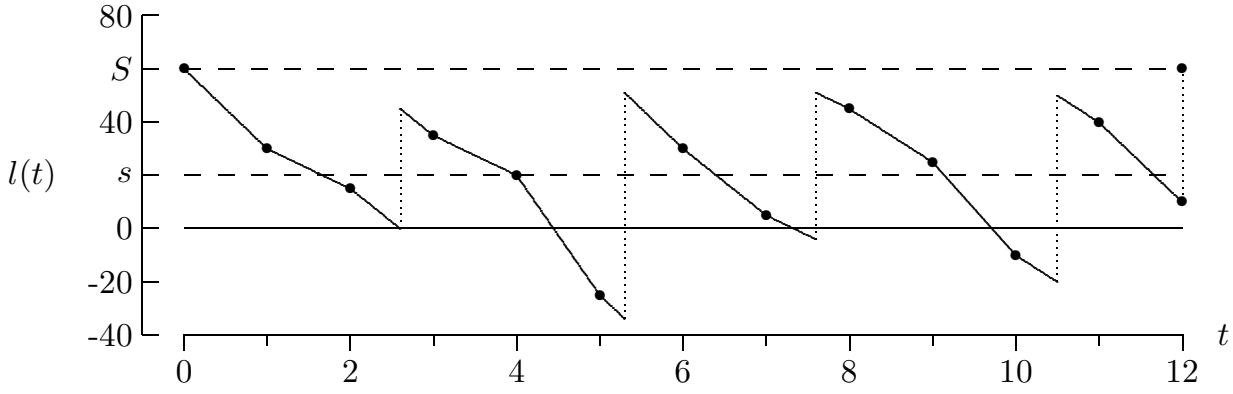


Figure: Inventory level *with* delivery lag

Time Evolution of Inventory Level

How should we revise *Algorithm 1.3.1*?

Algorithm 1.3.1

$l_0 = S;$
$i = 0;$
while (more demand to process) {
  $i++;$
  if $(l_{i-1} < s)$
    $o_{i-1} = S - l_{i-1};$
  else
    $o_{i-1} = 0;$
  $d_i = GetDemand();$
  $l_i = l_{i-1} + o_{i-1} - d_i;$
}
$n = i;$
$o_n = S - l_n$
$l_n = S;$
return l_1, l_2, l_3, ...l_n and o_1, o_2, ...o_n;

Statistical Considerations

- If $l_{i-1} \geq s$ the equations for $\bar{l}_i^+$ and $\bar{l}_i^-$ remain correct since there is no order, hence no delivery lag
- When delivery lag occurs the time-averaged holding and shortage intervals must be modified
  - The delivery lag for interval i is $0 < \delta_i < 1$

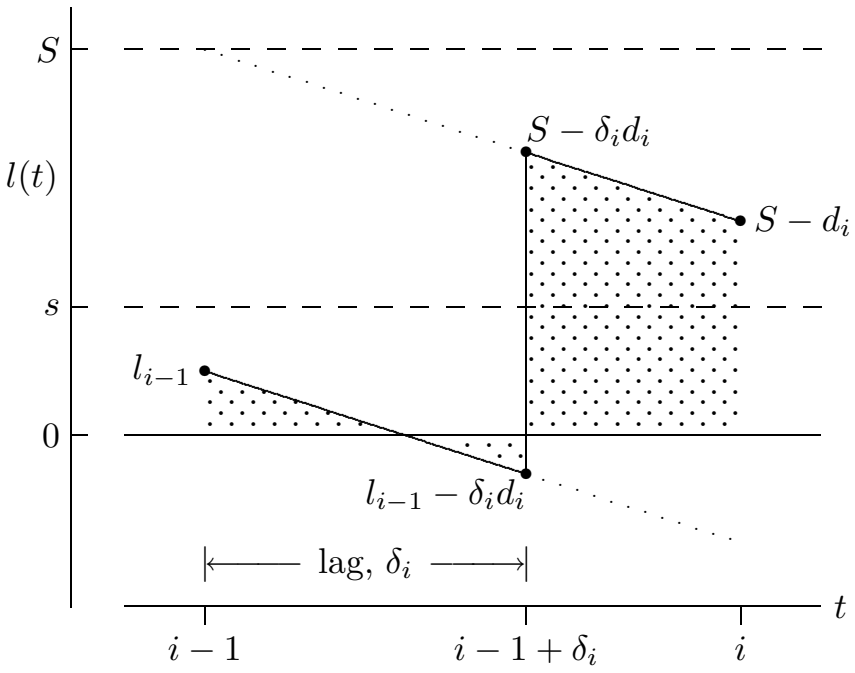


Figure: Inventory level *with* delivery lag

Time Evolution of Inventory Level

How should we revise *Algorithm 1.3.1*?

Algorithm 1.3.1

$l_0 = S$;
$i = 0$;
while (more demand to process) {
  $i++$;
  if ($l_{i-1} < s$)
    $o_{i-1} = S - l_{i-1}$; $\delta_i = GetDeliveryLag()$
  else
    $o_{i-1} = 0$;
  $d_i = GetDemand()$;
  /* compute statistics before the delivery arrives */
  .....
  /* compute statistics after the delivery arrives */
  .....
  $l_i = l_{i-1} + o_{i-1} - d_i$;
}
$n = i$;
$o_n = S - l_n$
$l_n = S$;
return l_1, l_2, l_3, ...l_n; o_1, o_2, ...o_n; and δ_1, δ_2, ...δ_n;

Revising Algorithm 1.3.1

- Determining delivery lag
  - Recall: lag is *assumed* to be random and independent of order size
- Computing statistics
  - Calculations of statistics remain the same: average order, average demand, and relative frequency of setups
  - Calculations of statistics need to change: time-averaged holding level, and time-averaged shortage level
    - How to calculate: view the calculation as estimating the area of trapezoids.

Consistency Checks

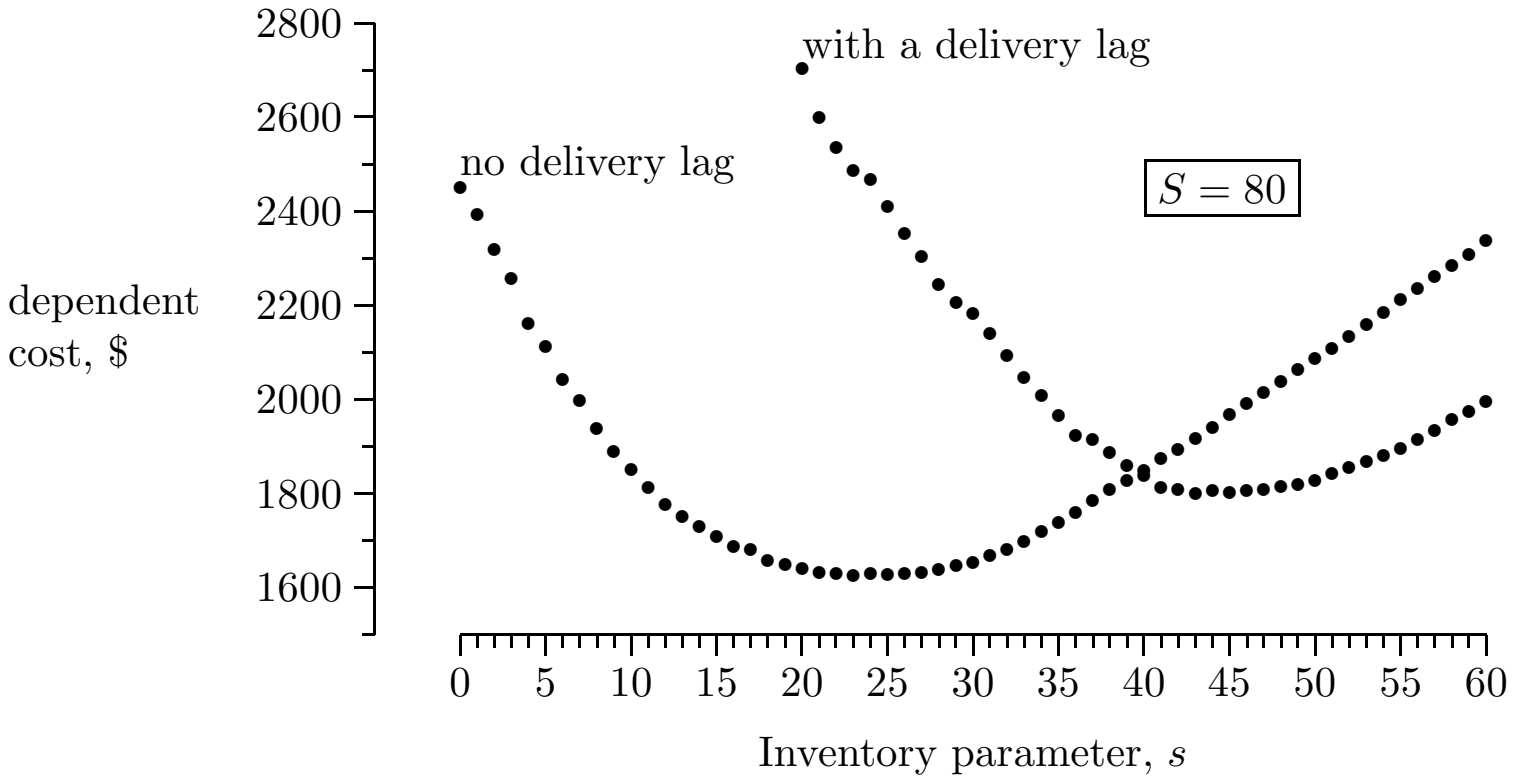
- It is fundamentally important to verify extended models with the parent model
  - Set system parameters to special values
- Set $\beta = 0$ for the SSQ with feedback
  - Verify that all statistics agree with parent
- Using the library rngs facilitates this kind of comparison
- It is a good practice to check for intuitive "small-perturbation" consistency
  - Use a small, but non-zero β and check that appropriate statistics are slightly larger

Example 3.3.3

- For the SIS with delivery lag, $\delta_i = 0.0$ if and only if no order during i-th interval, $0 < \delta_i < 1.0$;
- Otherwise, the SIS is lag-free if and only if $\delta_i = 0.0$ for all i
- If (S, s) are fixed then, even with small delivery lags:
  - $\overline{o}$, $\overline{d}$, and $\overline{u}$ are the same regardless of delivery lag
  - Compared to the lag-free system, $\overline{l}^+$ will decrease
  - Compared to the lag-free system, $\overline{l}$ will increase or remain unchanged

Example 3.3.4

- Delivery lags are independent Uniform(0.0, 1.0) random variates



  - Delivery lag causes $\overline{I}^{+}$ to decrease and $\overline{I}$ to increase or remain the same
  - $C_{Setup} = \$1,000$, $C_{hold} = \$25$ and $C_{short} = \$700$ cause it to shift up and to the left

Exercise L7-2

You will complete examples 3.3.3 and example 3.3.4, for which, you will revise *sis2* program.

- Make a copy of the *sis2* program. Name the copy as *sis2ex72* that you will be working on.
- Replace library *rng* by library *rngs*
- Use two random streams to generate demand and delivery lag
  - Revise *GetDemand()* in C or *getDemand()* in Java
  - Add a new function *GetDeliveryLag()* in C or *getDeliveryLag()* in Java
    - Use $Uniform(0, 1)$ random variate delivery lag.
- Calculate statistics: average order, average demand, relative frequency of setups, time-averaged holding level, and time-averaged shortage level, and dependent cost (i.e., the sum of the average setup, holding, and shortage costs)

Exercise L7-2: Recap

Complete the consistency checks as outlined in slides 22 and 23 using the instructor's sample solution.

- Compute statistics when the delivery lag is generated using random variates $Uniform(0, 0.00)$, $Uniform(0, 0.01)$, $Uniform(0, 0.02)$, ... and $Uniform(0, 0.10)$.
- Perform consistency checks:
  - Special value: we expect that the statistics should be the same as those obtained from *sis2* when the delivery lag is 0
  - Small-perturbation: when the delivery lag is small, we expect that the statistics should be different from those obtained from *sis2*, but the different should be small.
  - Intuition: if there is a delivery lag, we expect to observe that average holding should decrease and average shortage should increase when compared to the lag-free system; average order, average demand, and order frequency are independent of delivery lags.
- Use a well-designed graph to show the results of the consistency checks.

Single-Server Machine Shop

Let us consider a machine shop.

- It has a finite number of identical machines.
- Each machine operates continuously until failure.
- As machines fail, they are repaired, in the order in which they fail, by a service technician.
- As soon as a failed machine is repaired, it is put back into operation.

The machines when they are operate produce an income.

- e.g., a machine operates 8 hours per day, 250 days a year, and produces a net income of $20 per hour of operation.

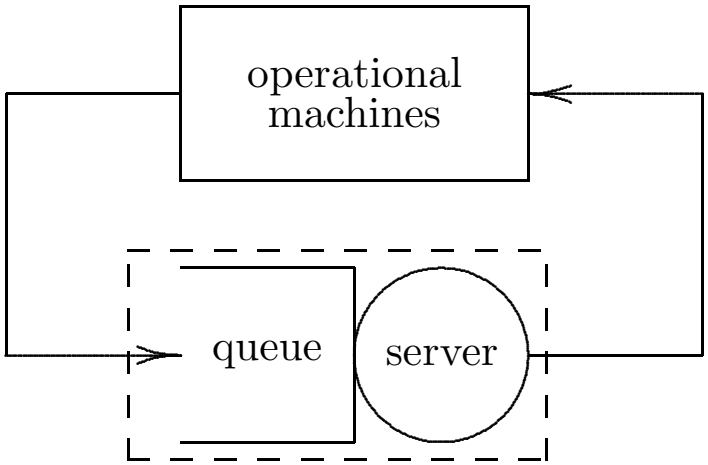
The service technician takes time to repair machine which leads to a cost.

- Hiring a technican costs money.
- However, more technicans would allow machines to be repaired more quickly and put back to operations, which leads to more income.

The *objective* is how one may maximize the profit by hiring the right number of technicans.

Single-Server Machine Shop: Specification

- The machine shop model is *closed* because there are a finite number of machines in the system.



- Assume repair times are $Uniform(1.0, 2.0)$ random variates
  - To reduce the average pair times, one may hire more technicians.
- There are M machines that fail after an $Exponential(100.0)$ random variate

Single-Server Machine Shop: Computational Model

Single-Server Machine Shop

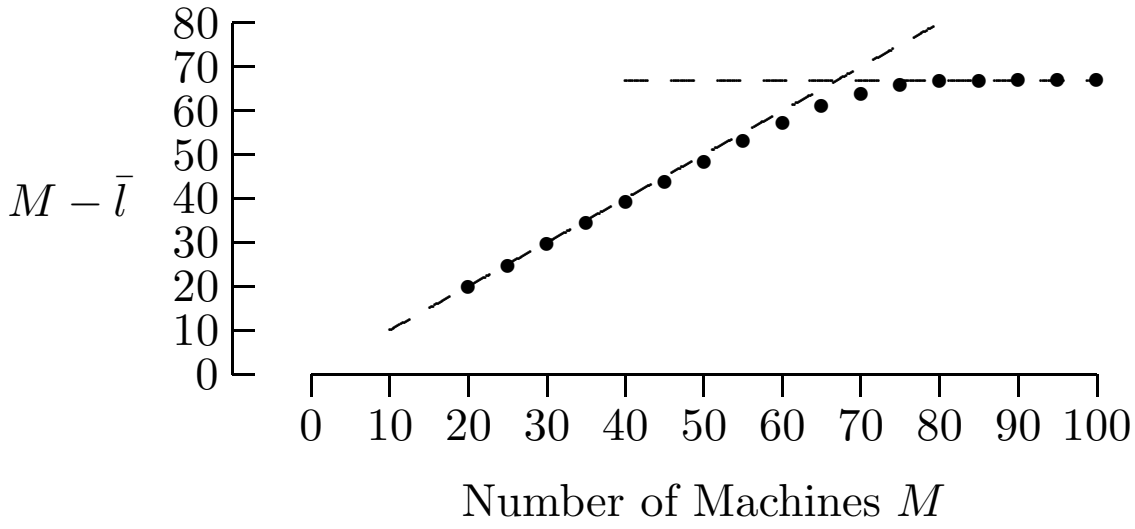
$C_0 = 0.0;$
$i = 0;$
for each machine i in $\{m_1, m_2, \ldots m_M\}$
  initialize f[i];
while (more failure to process) {
  i ++;
  $(a_i, m) = NextFailure();$
  if $(a_i < c_{i-1})$
    $d_i = c_{i-1} - a_i;$
  else
    $d_i = 0.0;$
  $s_i = GetSerivce();$
  $c_i = a_i + d_i + s_i;$
  $f[m] = c_i + GetFailure();$
}
$n = i;$
return $d_1, d_2, \ldots, d_n;$

Single-Server Machine Shop: Computational Model

- Program *ssms* simulates a single-server machine shop
- The library *rngs* is used to uncouple the random processes
- The failure process is defined by the array *failures*
  - A $\mathcal{O}(M)$ search is used to find the next failure
  - Alternate data structures can be used to increase computational efficiency

Example 3.3.5

- The time-averaged number of working machines is $M - \bar{l}$



- For small values of M the time-averaged number of operational machines is essentially M
- For large values of M this value is essentially constant at approximately 67

Exercise L7-3

Complete the following tasks using *ssms*

- Relative to Example 3.3.5 in slide 31, construct a figure illustrating how $\overline{x}$ (utilization) depends on M.
- If you extropolate linearly from small values of M, at what value of M will saturation ($\overline{x} = 1$) occur?
- Can you provide an empirical argument or equation to justify this value?

Summary

Three examples:

- SSQ with immediate feedback
- SIS with delivery lag
- Single-Server Machine shop