SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NY 14853-3801

TECHNICAL REPORT NO. 1045

May 1993

Delphi: A C-Based Queuing Network Simulator[1]

by

F. Chance

[1]Research supported by the Defense Advanced Research Projects Agency.

ABSTRACT

We describe Delphi, a package for simulating open queuing networks. Multiple product types with priorities, product scrap and rework, deterministic and probabilistic routing are supported. Queue control routines are separated from the main code, to facilitate their replacement by user routines.

Delphi was developed using the Sigma (Schruben 1991) simulation prototyping system, then converted into portable C. Delphi should compile and run on virtually any platform supporting a good subset of the standard C libraries. The source code and supporting documentation are available via anonymous ftp from 'gauss.orie.cornell.edu'. This document is current for Delphi Version 5, Level 7.

Contents

1 INTRODUCTION

Delphi was developed to test initialization bias truncation methods on large, realistic queuing networks. The inspiration for some of the features supported in Delphi is the semiconductor manufacturing simulation described in Hood et al. (1989), although in general Delphi models less process detail.

2 DOCUMENTATION PROCEDURE

In the belief that written documentation for an evolving program becomes obsolete as soon as it is printed, this document is meant to teach the basics about Delphi's operation, introduce the main command line options, and show how to construct and simulate many small building block models (single server queues, small networks, etc). This document is not meant to spell out the nuts and bolts of each procedure within the code. That documentation is left within the code itself.

3 OBTAINING AND COMPILING DELPHI

Delphi is available via anonymous ftp from 'gauss.orie.cornell.edu'. Anonymous ftp is a program available on most Unix platforms that allows access to a limited amount of data on another user's machine, without the need for guest passwords or special login accounts. Delphi comes packaged in a compressed **tar** file (that means you must uncompress and untar it after you ftp it.) **tar** is a command that either puts lots of little files into one big file, or the reverse. The following example shows how one can obtain and compile Delphi on a standard unix machine. Commands that you should type are displayed in **bold face**.

```
gauss% mkdir delphi
gauss% cd delphi
gauss% ftp gauss.orie.cornell.edu
Connected to gauss.orie.cornell.edu.
220 pivot FTP server (SunOS 4.1) ready.
Name (gauss.orie.cornell.edu:chance):  anonymous
331 Guest login ok, send ident as password.
Password:  chance@orie.cornell.edu
230 Guest login ok, access restrictions apply.
ftp> cd pub/delphi
250 CWD command successful.
```

```
ftp> binary
200 Type set to I.
ftp> get delphi.tar.Z
200 PORT command successful.
150 Binary data connection for delphi.tar.Z
226 Binary Transfer complete.
local:  delphi.tar.Z remote:  delphi.tar.Z
44516 bytes received in 0.3 seconds
ftp> quit
221 Goodbye.
gauss% uncompress -f delphi.tar.Z
gauss% tar -xvf delphi.tar
(Tar messages for each file that is extracted.)
gauss% make delphi
(Compiler messages for each file that is compiled.)
gauss% delphi
(Current program options displayed).
gauss%
```

If Delphi displays a list of options when invoked, it has successfully compiled. Otherwise, try to obtain help from a local guru, and check that you specified **binary** before transferring the file with ftp.

4 PRINTING DELPHI DOCUMENTATION

This document is included with the Delphi distribution in the file **delphi.tex**. This file is in LaTeX format, so if you have the LaTeX processor installed on your system, you can process and print this document. See Lamport (1986) for details of the LaTeX system.

If you do not have the package **psfig** installed on your system, you will not be able to print the event graph for Delphi. Figure 1 shows a typical sequence of commands used to process and print this document on a postscript printer if **psfig** is not installed. Figure 2 shows a typical sequence of commands on a system with **psfig** installed.

5 VERSION AND LEVEL NUMBERS

If Delphi is executed with no command line options, it displays a version and level number in addition to information about input file formats and available options. The version number

```
gizmo% nopsfig
psfig disabled
gizmo% texprep
Working...
Delphi tex file preparation complete
gizmo% latex delphi.tex
(Messages from Tex)
gizmo% latex delphi.tex
(Rerun to get references correct.)
gizmo% dvips delphi.dvi > delphi.ps
gizmo% lpr delphi.ps
```

Figure 1: Typical commands for processing documentation on a system without **psfig** and a postscript printer.

```
gizmo% psfig
psfig enabled
gizmo% texprep
Working...
Delphi tex file preparation complete
gizmo% latex delphi.tex
(Messages from Tex)
gizmo% latex delphi.tex
(Rerun to get references correct.)
gizmo% dvips delphi.dvi > delphi.ps
gizmo% lpr delphi.ps
```

Figure 2: Typical commands for processing documentation on a system with **psfig** and a postscript printer.

changes only when modifications are made so that Delphi is no longer reverse-compatible with the input file format from earlier versions. The level number is incremented whenever a major modification is completed. Level numbers restart at 0 whenever the version number is incremented. Thus, input files from earlier versions will not work and must be converted in some way, but those from the same version but an earlier level are still supported.

6 SIMULATING AN $M/M/1$ QUEUE

We begin with the canonical queuing theory example, the $M/M/1$ queue. Briefly, a single server (Lucy) dispenses service (psychiatric help) to customers (Charlie Brown, Linus, etc.) arriving at randomly spaced intervals. Interarrival times are independent, identically distributed (i.i.d.) exponential random variables with mean $1/\lambda$. Service times are i.i.d. exponential random variables with mean $1/\mu$. The server never becomes tired, requires sick leave, down time, coffee breaks, or football practice. We denote the *traffic intensity*, roughly a measure of how busy the server will be on a scale from 0 to 1, by $\rho = \lambda/\mu$. The system is said to be *stable* (number of customers waiting for help doesn't tend to infinity), if $\rho < 1$. We will often interchange the terms *job* and *customer*.

For Delphi to simulate this system, we need to tell it the following two types of information: system parameters (number of queues, interarrival distribution, etc) and routing parameters (where each job goes, service time distributions, etc). Suppose we name our simulation **mm1**. Delphi will look in the file **mm1.del** for system information, and in the file **mm1.r1** for routing information. For convenience, these files are included in the Delphi distribution package. We briefly outline what each contains.

The configuration file **mm1.del** is shown in Figure 3. The routing file **mm1.r1** is shown in Figure 4.

For our convenience, think of all time units as hours. Suppose the mean time between arrivals to Lucy's stand is one hour, and the average customer receives about 39 minutes ($0.65 * 1$ hour) of helpful advice. Here's what happens when we execute Delphi for 100 customers on our $M/M/1$ example.

```
gauss% delphi 1 mm1 -endjob 100

delphi v5.7: Starting simulation 'mm1'.
Successfully read in 1 routing entries, job type 1.
Number of tool groups/queues = 1
Number of replications = 1
Job type 1, priority 1, number of releases 1.
```

```
{   This File: mm1.del
      Creator: Frank Chance
Last Modified: 12-19-92
  Description: Input configuration file for DELPHI simulation
               of an M/M/1 queue.
}
{ Jobtype 1 description line. }
{ <Line ID> <jobtype> <priority> <#components>}
     j       Customers          1            1
{ Release patterns for jobtype 1. }
{ <Line ID> <#jobs to release>   <Inter-release distribution> }
     r                 1                        e(1.0)
{ Queue 1 description line. }
{ <Line ID> <Queue Name> <# Servers> <Server Type> }
     q         Lucy            1           5
```

Figure 3: Configuration file for $M/M/1$ queue.

```
{   This File: mm1.r1
      Creator: Frank Chance
Last Modified: 12-19-92
  Description: Routing configuration file for DELPHI
               simulation of an M/M/1 queue.
}
{ Routing step 1 description line.}
{ <Line ID> <Step> <Queue> <Load> <Process> <Unload> }
      p     Consult  lucy    c(0)   e(0.65)    c(0)
```

Figure 4: Routing file for $M/M/1$ queue.

```
Normal completion, replication 1 of 1, time = 104.707.
delphi v5.7: Simulation 'mm1' exiting normally.
```

Output statistics are contained in the file **mm1.run**. Use your favorite editor/browser to view this file. Most statistics are self-explanatory.

To check that simulation does eventually match analytic results, try running the simulation for 10,000 jobs. In our case, $\lambda = 1$, $\mu = 1/0.65$, and $\rho = 0.65$. The limiting average time in system is $1/(\mu - \lambda)$ which in our case works out to be approximately 1.857. As you run for longer and longer periods, the average time in system should converge to 1.857. If the rate of convergence seems frighteningly slow, this should serve as a lesson in the perils of simulation. Large amounts of computer time may be necessary to reach accurate conclusions about simulated systems, even such simple ones as Lucy's psychiatric service center.

7 EVENT STRUCTURE OF DELPHI

Delphi was originally modeled using the Sigma simulation prototyping system, and thus has an underlying event graph structure. See the Sigma (Schruben 1991) documentation for a more complete explanation of event graphs. The event graph for Delphi is shown in Figure 5.

Nodes in the graph represent events, such as job releases. Arcs are shown where the execution of an event may schedule one or more future events. Conditions on the scheduling of such events are numbered and described. When an event schedules a future event, there may be a time delay. These time delays are lettered and described.

8 JOB TYPE, QUEUE, AND STEP NAMES

In the Delphi configuration and routing files, names are assigned for job types, queues, and process steps. In the example from the previous section the single job type was 'Customers', the queue name was 'Lucy', and the process step was 'Consult'. This section describes the way Delphi handles these names.

First, names are separated from the other information in the input files by spaces, so names cannot contain spaces, tabs, or new-lines. Second, names may be any combination of non-space printable characters up to ten digits long. Only the first ten digits will be used on reports, and in any internal comparisons.

Third, names are not case sensitive. Thus, 'Lucy' and 'lucy' both specify the same queue.

Finally, you cannot have two job types, queues, or process steps with the same name. You could have a queue and a process step with the same name, and Delphi would not be

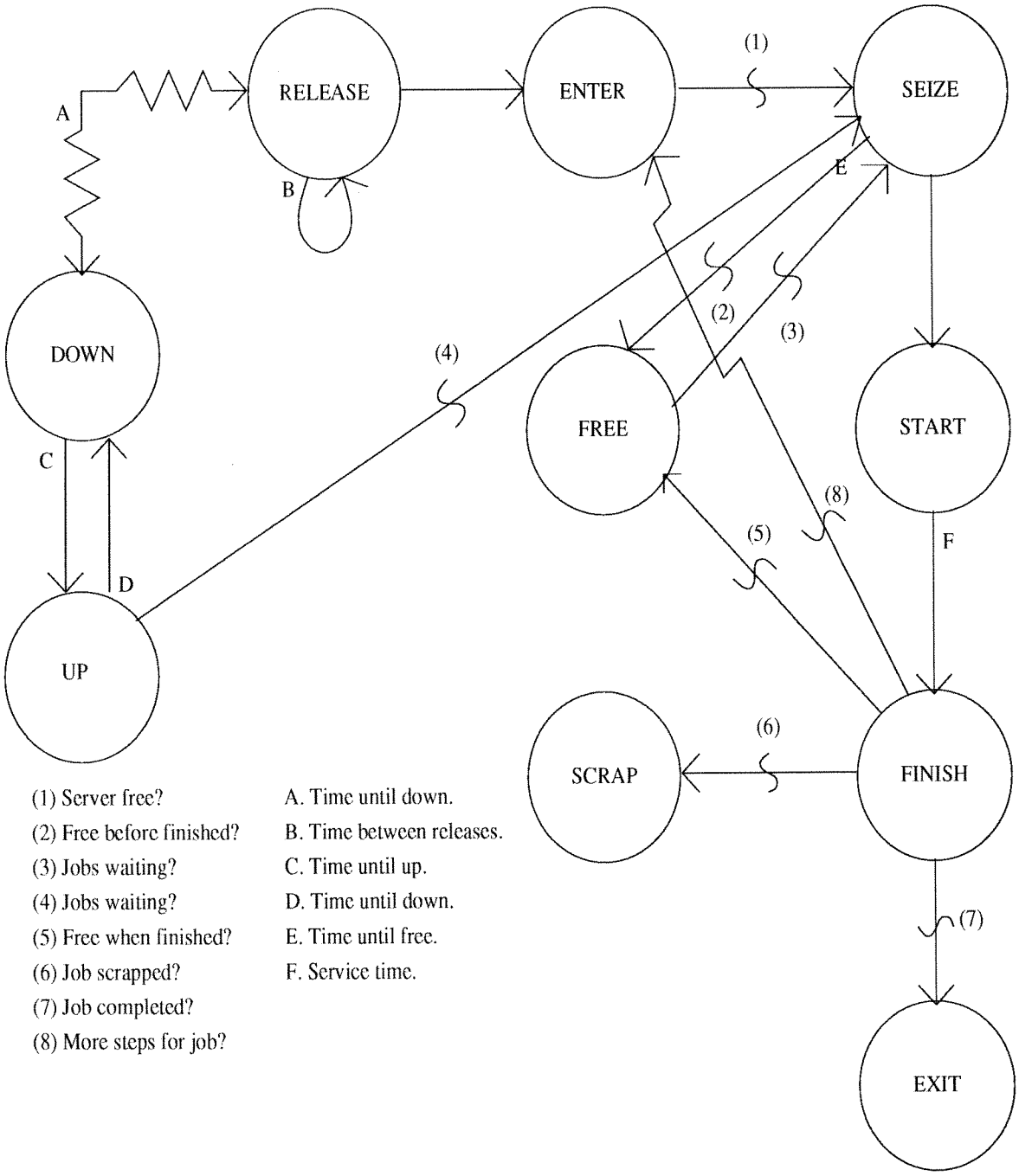

Figure 5: Delphi event graph.

Server Type	Description	Number of Required Parameters
1	Batch	3
2	Single-Component	3
3	Conveyor	4
4	Multi-sequence	4
5	Whole Job	3

Table 1: Server types and required parameters.

hindered, but it might be confusing to humans, so it is probably best to make names unique across all job types, queues, and process steps.

9 SERVER TYPES AND SERVICE TIME PARAMETERS

Five different types of servers are modeled within Delphi. The server type for a queue is identified in the configuration file **config.del**. In the routing file **config.r***j*, the correct number of service time parameters must be specified on the processing line. The server types and number of parameters needed are listed in Table 1.

Batch servers require three parameters: load time, batch processing time, and unload time. After appropriate batching has occurred, the service time for a batch is given by the sum of the three times.

Single-component servers require three parameters: load time, processing time per component, and unload time. The service time for a job is the load time plus processing time per component * number of components plus unload time. If the processing time per component is a random variable, Delphi will try to compute the distribution of the sum and generate from that distribution. If that is not possible, it will generate a processing time for each component and add these together.

Conveyor servers require four parameters: load time, the time for the first component to complete service, the inter-departure time for subsequent components, and unload time. The service time for a job is the load time plus the first component time, plus (number of components-1) times the inter-departure time, plus the unload time.

Multisequence servers require four parameters: load time, total time, longest time, and unload time. The service time for a job is as for a batch tool, with the processing time being the total time. However, the server becomes free before the completion of the service time. Delphi models this type of tool by generating a random variable from the total time

Distribution	Description
c(value)	Constant.
e(mean)	Exponential.
u(a,b)	Uniform between a and b.
tp(mean,pct)	Triangular, mean +/- pct%.
up(mean,pct)	Uniform, mean +/- pct%.

Table 2: Distributions supported in Delphi.

distribution, then multiplying this realization by the ratio of the mean of the longest time to the mean of the total time. The tool is freed after the load time plus this calculated time has passed.

Per-job servers require three parameters: load time, processing time for the job, and unload time. The service time for a job is the sum of these three times.

10 SPECIFYING DISTRIBUTIONS

Service times and interarrival times in Delphi can be specified as random variables. In general, a distribution is specified in the input file in the form **dist(parms)**. The distributions currently supported appear in Table 2.

11 ADDING DOWN-TIME TO A MODEL

Suppose that every 8 hours, Lucy takes a break that lasts for an exponentially distributed amount of time, one hour being the mean time away from her duties. The modifications necessary to our earlier $M/M/1$ model are included in the files **cf-mm1.del** and **cf-mm1.r1**. Actually, **cf-mm1.r1** is exactly the same as **mm1.r1**, but for simplicity, Delphi requires that system configuration and routing configuration files use the same prefix.

The file **cf-mm1.del** is marginally more complicated than before, and is displayed in Figure 6.

We can try out our new model for ten days by using the command line option **-endtime**, as follows.

```
gauss% delphi 1 cf-mm1 -endtime 240

delphi v5.7: Starting simulation 'cf-mm1'.
Successfully read in 1 routing entries, job type 1.
```

```
{   This File: cf-mm1.del
      Creator: Frank Chance
Last Modified: 12-19-92
  Description: Input configuration file for DELPHI simulation
               of an M/M/1 queue with clock-time failures.
}
{ Jobtype 1 description line. }
{ <Line ID> <jobtype> <priority> <#components>}
     j       Customer      1             1
{ Release patterns for jobtype 1. }
{ <Line ID> <#jobs to release>   <Inter-release distribution> }
     r                 1                        e(1.0)
{ Queue 1 description line. }
{ <Line ID> <Queue> <# Servers> <Server Type> }
     q       Lucy         1           5
{ Queue 1 down-time line. }
{ <Line ID> <Time-to dist> <Time-offline dist> }
     cf          c(1.0)            e(1.0)
```

Figure 6: Configuration file for $M/M/1$ queue with breakdowns.

```
Number of tool groups/queues = 1
Number of replications = 1
Job type 1, priority 1, number of releases 1.
Normal completion, replication 1 of 1, time = 240.
delphi v5.7: Simulation 'cf-mm1' exiting normally.
```

As before, detailed output statistics appear in the file **cf-mm1.run**. Down-times within Delphi can be modeled in one of three ways. The first two are closely related, and we will denote them by "clock-time" and "busy-time" down-times. For both of these types, a time-to distribution and a time-offline distribution are specified. First, a variate is generated from the time-to distribution. For clock-time down-times, the down-time occurs when this amount of simulated time has passed. For busy-time down-times, the down-time occurs when the server has been busy processing jobs for this amount of time.

When the down-time occurs, a variate is generated from the time-offline distribution, and the server is down for that amount of time. The process is then restarted with the generation of another time-to variate.

Within either clock-time or busy-time down-times, failures, preventive maintenance, and setups can be simulated. To model a clock-time or busy-time down-time, add an extra line after the queuing description line in the input configuration file, specifying the time-to and time-offline distributions. For explicit format instructions, see the output when Delphi is executed without any command line options.

A third way of modeling breakdowns is to model them as a jobtype that crosses only a single machine, and has the highest priority of any job. Under this assumption, breakdowns can queue at a machine, certainly not an intuitive situation. It is possible to model this assumption in Delphi, however, by creating an extra jobtype for each machine with this type of down-time.

12 ADDING SCRAP TO A MODEL

Suppose Lucy decides to branch out of the psychiatric counseling business into football manufacturing. The raw materials for her operation are the leather and the laces, her operation being to simply insert the laces into the pigskin. Boxes of 10 footballs each arrive via the postal service at randomly space intervals (exponentially distributed interarrival times with mean 8 hours). Suppose further that Lucy is a stickler for quality, and enforces rather strict rules about what can and cannot be sent out the door to the local football retailers.

Being a small time operator, her entire operation fits under the tree in the backyard, which is fine if it doesn't rain that day, but bad news if it rains, because the laces shrink up and don't fit properly. Furthermore, there is a small chance that when she is lacing the footballs, she inadvertently sticks the ball with her lacing needle, and the football is history.

Let's say that from Lucy's previous job experience as a weather forecaster, she knows that one the average, one out of every twenty days is rainy (she obviously does not operate in Ithaca). Also, from a few days worth of experience with lacing, she has gotten pretty good, and she only punctures about every one in ten footballs.

The file **s-mm1.del** is shown in Figure 7.

```
{   This File: s-mm1.del
      Creator: Frank Chance
Last Modified: 12-19-92
  Description: Input configuration file for DELPHI simulation
               of an M/M/1 queue with scrap.
}
{ Jobtype 1 description line. }
{ <Line ID> <jobtype> <priority> <#components> }
     j       Footballs     1           10
{ Release patterns for jobtype 1. }
{ <Line ID> <#jobs to release>   <Inter-release distribution> }
     r                 1                        e(8.0)
{ Queue 1 description line. }
{ <Line ID> <Queue>   <# Servers> <Server Type> }
     q       Lacing         1           2
```

Figure 7: Configuration file for $M/M/1$ queue with scrap.

The routing file **s-mm1.r1** is shown in Figure 8.

Note that the specification of scrap in Delphi is more general than the situation we have described here. The three scrap parameters are

$$100.0 * P[\text{scrap occurs}],$$
$$100.0 * P[\text{entire job scrapped}|\text{scrap occurs}],$$
$$100.0 * P[\text{component scrapped}|\text{scrap occurs, entire job not scrapped}].$$

We can handle our situation described above by setting the first scrap parameter to 100.0.

```
{   This File: s-mm1.r1
      Creator: Frank Chance
Last Modified: 12-19-92
  Description: Routing configuration file for DELPHI
               simulation of an M/M/1 queue with scrap.
}
{ Routing step 1 description line.}
{ <Line ID> <Step>    <Queue>  <Load> <Process> <Unload> }
      p      Lacing    Lacing   c(0)   e(0.50)    c(0)
{ Scrap description line.}
{ <Line ID> <% Scrap Occurs> <Job %> <Component %> }
      s          100.0         5.0        10.0
```

Figure 8: Routing file for $M/M/1$ queue with scrap.

We execute the model exactly as before. Output statistics for scrap are included in **s-mm1.run**.

13 ADDING REWORK TO A MODEL

Lucy's football manufacturing business expands at a phenomenal rate, and before long, she is taking orders for private label footballs from the NFL, K-mart, and the White House. She hires one of the neighbor kids to stencil on the private label after the lacing operation is complete. She also wants to cut out scrap completely, so after each box of footballs has completed the lacing step, she has someone standing by with some puncture-fixing glue. Any football that was punctured in the lacing step gets a squirt of glue, then the entire box is returned to the lacing step, where laces are put into the previously punctured (but now good as new) footballs.

Approximately one in ten footballs is punctured (Lucy's success rate with the lacing procedure hasn't increased.) The puncture-fixing operation takes a constant six minutes to apply per football. To lessen confusion in the workplace, the footballs remain in their boxes of 10, so when any footballs in a box are reworked, the entire box cannot proceed, until all are ok.

After a box is successfully laced, it proceeds to the private label attacher. This operation takes a constant 12 minutes per football (0.2 hours).

Finally, the box is closed up and Lucy's patented "seal of quality" is put across the top. This operation takes a constant 6 minutes per job.

The file **r-mm1.del** is shown in Figure 9.

```
{   This File: r-mm1.del
      Creator: Frank Chance
Last Modified: 12-19-92
  Description: Input configuration file for DELPHI simulation
               of a small system with rework and scrap.
}
{ Jobtype 1 description line. }
{ <Line ID> <jobtype> <priority> <#components> }
     j       Footballs      1           10
{ Release patterns for jobtype 1. }
{ <Line ID> <#jobs to release>   <Inter-release distribution> }
     r                 1                        e(8.0)
{ Queue description lines. }
{ <Line ID> <Queue Name> <# Servers> <Server Type> }
     q         Lacing          1           2
     q         Fixup           1           2
     q         Add-labels      1           2
     q         Seal-box        1           5
```

Figure 9: Configuration file for system with rework and scrap.

The routing file **r-mm1.r1** is shown in Figure 10.

Rework percentages are specified much like scrap:

$$100.0 * P[\text{rework occurs}],$$
$$100.0 * P[\text{entire job reworked}|\text{rework occurs}],$$
$$100.0 * P[\text{component reworked}|\text{rework occurs, entire job not reworked}].$$

Rework statistics are included in the output file **r-mm1.run**.

```
{   This File: r-mm1.r1
      Creator: Frank Chance
Last Modified: 12-19-92
  Description: Routing configuration file for DELPHI
               simulation of small system with
               rework and scrap.
}
{ <Line ID>  <Step>  <Queue> <Load> <Process> <Unload> }
      p      Lacing   Lacing  c(0)   e(0.50)    c(0)
{ <Line ID> <% Scrap Occurs> <Job %> <Component %> }
      s          100.0         5.0        10.0
{ <Line ID> <Skip-to-step> <% Rework occurs> <Job %> <Comp %>}
      r       Add-labels         100.0        0.0      10.0
{ <Line ID>     <Step>          <Queue>     <Load> <Process> <Unload> }
      p       Fixup             Fixup   c(0)   c(0.10)    c(0)
{ <Line ID> <Goto-line> <% of time>}
      g       Lacing          100.0
{ <Line ID>    <Step>           <Queue>      <Load> <Process> <Unload> }
      p       Add-labels      Add-labels      c(0)   c(0.20)    c(0)
      p       Seal-box        Seal-box        c(0)   c(0.10)    c(0)
```

Figure 10: Routing file for system with rework and scrap.

14 EXAMPLE OUTPUT STATISTICS

A variety of output statistics are contained in the file **config.run**. We will consider an output file for a five thousand job run of the example discussed in the previous section, **r-mm1.run**. This output file is displayed in Figures 11 through 17.

15 PROBABILISTIC ROUTING

In order for Delphi to be able to handle general networks of queues with Markov routing, there must be the capability after each processing step at a queue to jump randomly to any queue within the system. This capability is given by the 'g' step in the routing file.

As an example consider a simple three step operation, where the second step is only performed about one-third of the time. The files **g-mm1.del** and **g-mm1.r1** are included in the Delphi distribution and implement such a system.

The routing file **g-mm1.r1** is shown in Figure 18, along with explanations of each line.

When this sytem is executed, all jobs are processed at queue 1. After being processed at queue 1, jobs are routed directly to queue 3 with probability 0.33, otherwise they continue on to queue 2 with probability 0.67. All jobs are processed at queue 3.

16 CALCULATING CONFIDENCE INTERVALS

Delphi uses two different methods to calculate confidence intervals for time in system and time at individual queues.

First, to calculate confidence intervals for time at a queue, Delphi uses the queuing relation $L = \lambda W$, where L is the expected value of the limiting number at the queue, λ is the arrival rate into the queue, and W is the expected value of the limiting time in queue. If we let $Q(t)$ be the number at the queue (waiting plus being served) at time t and the simulated clock runs from time 0 to time T, we can estimate L by

$$\hat{L} = \frac{1}{T}\int_0^T Q(t)dt.$$

If we know λ, or can estimate it by $\hat{\lambda}$, we estimate W by

$$\hat{W} = \frac{\hat{L}}{\hat{\lambda}},$$

although the exact effect of dividing by an estimate of λ rather than the true λ is unknown.

```
---- End-of-Replication Report, (Replication 1 of 1) ---

Run ended at simulated time: 45295.4
Delphi Version 5, Level 7.
Runlist file: r-mm1.run
Streams in use: 1 to 8.
Bytes of memory used        : 23288
# Calls to memory allocation: 147
# Times memory re-allocated : 26
Number of batches           : 20
Batch length (in time units)  : 2000
Statistics collected after time: 0

--- Initialization Bias Test ---

Schruben et al. (1983) 2-sided test for initialization
bias (see Operations Research 31: 1090-1108)

Note: P-values below 0.10 are evidence of
statistically significant differences between
batch means in the output.

          Schruben's      Degrees
Jobtype    Statistic   of Freedom   P-value
-----------------------------------------
Footballs       1.80           21      0.09
```

Figure 11: General information and bias test, 5000 job run of **r-mm1**.

```
--- Job Type Statistics ---

Note: Upper and lower bounds are 95% confidence bounds.

---- Job Type ----  ----------------- Time in System ----------------
         Number of  ---------- Mean -------   ------- Variance ------
          Finished    Lower           Upper     Lower           Upper
Name          Jobs    Bound    Mean   Bound     Bound     Var   Bound
-------------------------------------------------------------------------
Footballs     5000 (   16.8,   17.9,   19.1) (  116.5,  149.9,  183.2)

            ------------ Throughput Per Time Unit -----------
            ------ Components -----   --------- Jobs --------
Job Type      Lower           Upper     Lower           Upper
  Name        Bound    Mean   Bound     Bound    Mean   Bound
-------------------------------------------------------------------
Footballs  (  0.964,  0.990,  1.016) (  0.108,  0.110,  0.113)

--- Queue Statistics ---


                                  Max   ---- Queue Delay Only ---
              %     %     %       in      Lower              Upper
Queue      Work  Down  Free    Queue      Bound     Mean     Bound  Rank
-------------------------------------------------------------------------
Lacing     69.6   0.0  30.4    13 (      5.42,     6.20,     6.98)     1
Fixup       1.2   0.0  98.8     2 (      0.00,     0.00,     0.00)     3
Add-labels 19.8   0.0  80.2     4 (      0.31,     0.34,     0.38)     2
Seal-box    1.1   0.0  98.9     1 (      0.00,     0.00,     0.00)     4
```

Figure 12: System and queue delay statistics, 5000 job run of **r-mm1**.

```
--- Estimated Time in Queue Plus Service by Job ---

           Estimated     - Time in Queue+Service -                 Rank
           Number of       Lower                  Upper    % of    Within
Queue        Visits        Bound      Mean        Bound    Total   Jobtype
-------------------------------------------------------------------------
Job Type: Footballs
Lacing         1.66  (    14.33,     15.75,     17.17)   87.10         1
Fixup          0.66  (     0.09,      0.09,      0.10)    0.52         3
Add-labels     1.00  (     2.06,      2.14,      2.21)   11.82         2
Seal-box       1.00  (     0.10,      0.10,      0.10)    0.55         3
                                    --------
          Job Type Total:             18.08
```

Figure 13: Time in queue plus service for job, 5000 job run of **r-mm1**.

To arrive at a confidence interval for our estimate of time at queue, we split time into a series of equal length batches. Suppose the number of batches is N, and the length of each batch is T_B. We let

$$\hat{L_n} = \frac{1}{T_B}\int_{(n-1)T_b}^{nT_B} Q(t)dt$$

be the batch mean estimate of L from the nth batch.

Unless otherwise specified, Delphi splits the output for each job type into a default number of batches, and assumes the batch means are approximately independent, identically distributed normal random variables with unknown mean and variance. Under these assumptions, a t-confidence interval is placed about the average of the batch means.

Let $\hat{G}$ be the grand batch mean

$$\hat{G} = \frac{1}{N}\sum_{n=1}^{N}\hat{L}_n.$$

Let s_G^2 be the estimated variance of $\hat{G}$,

$$s_G^2 = \frac{1}{N-1}\sum_{n=1}^{N}(\hat{L}_n - \hat{G})^2.$$

--- Random Routing Statistics ---

Routing Step	To Route Step	#Random Choices	Out of Total	% Time Chosen
Job Type: Footballs				
Fixup	Lacing	3809	3809	100.00

--- Scrap Statistics ---

Routing Step	Number Visits	% Time Scrap Occurred	% Jobs Completely Scrapped	% Components Scrapped
Job Type: Footballs				
Lacing	9540	100.00	4.97	9.85

--- Rework Statistics ---

Routing Step	Number Visits	% Time Rework Occurred	% Jobs Completely Reworked	% Components Reworked
Job Type: Footballs				
Lacing	9540	92.34	0.00	10.17

Figure 14: Rework and scrap statistics, 5000 job run of **r-mm1**.

--- Number in System Statistics ---

All Job Types:

# in System	# of Hits	Total Time	% Time	Cumulative Percent
0	1349	10598.5885	23.40	23.40
1	2632	10304.4951	22.75	46.15
2	2302	8012.0203	17.69	63.84
3	1743	5424.4724	11.98	75.81
4	1248	3871.1271	8.55	84.36
5	881	2808.5863	6.20	90.56
6	579	1905.8735	4.21	94.77
7	352	1136.6482	2.51	97.28
8	199	626.0743	1.38	98.66
9	105	360.6566	0.80	99.46
10	47	134.8675	0.30	99.75
11	17	63.0146	0.14	99.89
12	8	37.5747	0.08	99.97
13	2	11.3547	0.03	100.00
		45295.3539		

* *

Figure 15: Number in system statistics, 5000 job run of **r-mm1**.

```
---- End-of-Run Report (1 Replication(s).) ---

Delphi Version 5, Level 7.
Runlist file: r-mm1.run
Streams in use: 1 to 8.
Bytes of memory used        : 23288
# Calls to memory allocation: 147
# Times memory re-allocated : 26
Number of batches             : 20
Batch length (in time units)  : 2000
Statistics collected after time: 0

--- Job Type Information ---

                     Constant    Number    Release   Number   Due-Date
Job Type    Priority  W.I.P.    Release       Wrap  Routing     Offset
  Name      Size      Policy?  Patterns     Policy  Entries       Time
-------------------------------------------------------------------------
Footballs    10    1     No         1    Startover        4        N/A

--- Queue Information ---

Note: c = Clock-time, b = Busy-time, A = Both.
              S
           F P e
           a . t Servers
           i M u   for   Service
Queue      l . p  Queue  Discipline
-----------------------------------
Lacing                1   FIFO
Fixup                 1   FIFO
Add-labels            1   FIFO
Seal-box              1   FIFO
```

Figure 16: End-of-run report, 5000 job run of **r-mm1**.

```
--- Command Line Options ---

Option      Enabled Type  Argument    Description
-----------------------------------------------
-debug        No   Flag               Debug trace.
-debugstart   No   Value              Starting time for debug trace.
-nistrace     No   Flag               Number in system trace.
-xtrace       No   Flag               Time in system trace.
-debugjob     No   Value              Specify job for debug trace.
-endtime      No   Value              Ending time of simulation.
-endjob       Yes  Value  5000.00     End run after number of jobs.
-endjobtype   Yes  Value  1.00        Jobtype for ending job.
-norunlist    No   Flag               Do not generate run output.
-reps         Yes  Value  1.00        Number of replications.
-msetrunc     No   Flag               Estimate mean-square-errors.
-pointbias    No   Flag               Estimate bias for individual jobs.
-nbatch       Yes  Value  20.00       Number of batches.
-cilevel      Yes  Value  95.00       Confidence interval level.
-saveint      No   Value              Interval between state-saves.
-readstate    No   String             Read initial state space.
-savestate    No   Flag               Save state-space at run end.
-xbtrace      No   Flag               Time in system trace-batch means.
-norelease    No   Flag               Do not release jobs into system.
-debugq       No   String             Specify queue for debug trace.
-repfiles     No   Flag               Replic. traces to separate files.
-debugev      No   String             Specify event for debug trace.
-nohead       No   Flag               No column headings in output.
-clearstats   Yes  Value  0.00        Clear statistics at this time.
-verify       No   Flag               Print stats for easy verification.
-jtnis        No   Flag               Print jobtype # in system stats.
-pointwait    No   Flag               Use waiting times for pointbias.
```

Figure 17: Delphi option settings, 5000 job run of **r-mm1**.

```
{   This File: g-mm1.r1
      Creator: Frank Chance
Last Modified: 11-14-92
  Description: Routing configuration file for DELPHI
               simulation of a small network.
}
{ Routing step 1 description line.}
{ <Line ID> <Step> <Queue>   <Load> <Process> <Unload> }
      p      Step-1 Queue-1   c(0)   e(0.65)    c(0)
{ Goto description line.}
{ <Line ID> <Goto-line> <% of time>}
      g      Step-3         33.0
{ Routing step 2 description line.}
{ <Line ID> <Step> <Queue>   <Load> <Process> <Unload> }
      p      Step-2 Queue-2   c(0)   e(0.65)    c(0)
{ Routing step 3 description line.}
{ <Line ID> <Step> <Queue>   <Load> <Process> <Unload> }
      p      Step-3 Queue-3   c(0)   e(0.65)    c(0)
```

Figure 18: Routing file for three machine network with probabilistic routing.

Then the approximate 95% confidence interval for the expected number at queue is

$$\hat{G} \pm t_{N-1}(0.05)\frac{s_G}{\sqrt{N}},$$

where $t_N(c)$ is the $(1-c)/2$ quantile of the t-distribution with N degrees of freedom.

Proceeding boldly, we use

$$\frac{\hat{G}}{\hat{\lambda}} \pm t_{N-1}(0.05)\frac{s_G}{\sqrt{N}\hat{\lambda}}$$

as our confidence interval for estimated time at queue.

To estimate time in system, Delphi uses transaction observations rather than the relation $L = \lambda W$. The reason for this choice is that scrapping is possible, and if we were to use $L = \lambda W$, we would be estimating the time in system for all jobs, including those that are scrapped. Usually, we only wish to estimate time in system for those jobs that exit normally after having complete all their steps.

Time is batched as before. Denote by $\hat{W}_n$ the average time in system of jobs exiting during the nth batch. Let $\hat{G}$ be the grand batch mean

$$\hat{G} = \frac{1}{N}\sum_{n=1}^{N}\hat{W}_n.$$

Let s_G^2 be the estimated variance of $\hat{G}$,

$$s_G^2 = \frac{1}{N-1}\sum_{n=1}^{N}(\hat{W}_n - \hat{G})^2.$$

Then the approximate 95% confidence interval for the expected time in system is

$$\hat{G} \pm t_{N-1}(0.05)\frac{s_G}{\sqrt{N}}.$$

Confidence intervals for the variance of the time in system distribution are calculated in a similar fashion, using batch estimates of the variance.

If **-endtime** is specified, the batch size is calculated as

$$\text{batch size} = \frac{\text{endtime}}{\text{number batches}}.$$

If **-endjob** is specified, the batch size is calculated as

$$\text{batch size} = \frac{\text{endjob} * \text{mean-time-between-arrivals[endjobtype]}}{\text{number batches}}.$$

The default number of batches is small, on the conservative side, so the default batch length is large. It may be possible to increase the number of batches with the **-nbatch** command line option, and hence lessen the width of the confidence interval, without lowering the probability that the confidence interval does cover the true expected value.

The confidence level, unless specified with the **-cilevel** option, defaults to 95%. That is, if all the assumptions about the independence and normality of the data were satisfied, the confidence intervals reported would cover the true values approximately 95% of the time. It is very important to note that this coverage level is appropriate only for examining confidence intervals one at a time. If multiple confidence intervals are examined, the probability that all hold simultaneously is at least

$$1 - (1 - \text{cilevel}) * \text{number of intervals examined.}$$

Quantiles of the t-distribution are obtained from C versions of the routines *VSTUD* and *VNORM* given in Bratley, Fox and Schrage (1987).

17 TESTING FOR INITIALIZATION BIAS

When using Delphi to estimate *steady-state* performance parameters of a system (long-run performance, independent of any set of initial conditions), it is necessary to consider the effects of initialization bias. That is, the distribution of the time in system for jobs early on in the simulation run does not match those later in the run. For example, if the model is started with no jobs present, early jobs will see very little congestion, and hence will have lower mean time in system, compared to later jobs.

Plotting the time in system observations, and averaging across replications as described in Welch (1983), is often a good way to identify the effect of initial conditions on the output. Another way is to test the hypothesis that the time in system batch means are all statistically equal. One test of this hypothesis is detailed in Schruben et. al. (1983), and is implemented in Delphi. Briefly, the test involves forming a test statistic that is sensitive to changes in the batch means, and which converges in distribution to a known form.

Let $\hat{W}_n$ be the average time in system for jobs in the nth batch, N the number of batches, $\hat{s}_G^2$ the estimated variance of the grand batch mean, S_k the cumulative sum of batch means,

$$S_k = \sum_{j=1}^{k} \hat{W}_n,$$

and $\bar{Y}_k$ the mean of the first k batches,

$$\bar{Y}_k = \frac{S_k}{k}.$$

The test statistic given in Schruben et. al. (1983) is

$$\hat{T} = \frac{\sqrt{45}}{N^{3/2}\hat{s_G}} \sum_{k=1}^{N} \left(1 - \frac{k}{N}\right) k(\bar{Y}_N - \bar{Y}_k).$$

In this form, the calculation of $\hat{T}$ requires keeping all the batch means until the end of the run. However, with some algebraic manipulation, we can use the alternate form

$$\hat{T} = \frac{\sqrt{45}}{N^{3/2}\hat{s_G}} \left(\bar{Y}_N \frac{(N+1)(N-1)}{6} - \sum_{k=1}^{N} S_k + \frac{1}{N}\sum_{k=1}^{N} kS_k\right),$$

which can be calculated as the simulation runs. Since we perform a 2-sided test (to protect against both positive and negative initial bias), Delphi uses the absolute value of this test statistic, $|\hat{T}|$. To compute the p-value of the statistic, Delphi calculates the area outside $-|\hat{T}|$ and $|\hat{T}|$ for the t distribution with $N-1$ degrees of freedom. Since the t distribution is symmetric, this amounts to calculating

$$2(1 - F_{n-1}(|\hat{T}|)),$$

where $F_n(x)$ is the distribution function for the t distribution with n degrees of freedom.

The p-value of the statistic is the lowest significance level at which we would reject the hypothesis that the batch means are equal. The lower the p-value, the more evidence we have for the existence of statistically significant differences among the batch means.

Since the formation of confidence intervals for the mean time in system depends on identically distributed batch means, the run length should be increased until the p-value for the initialization bias test falls below a reasonable level, say 0.10, before other statistics given in the output file can be viewed with any confidence.

18 COMMONLY USED OPTIONS

The descriptions in this section are valid for the version and level of Delphi given in the abstract. Delphi requires at least the following basic information: a beginning random number stream, an input file prefix, and the length of the simulation run. Above we used the following command to simulate our simple $M/M/1$ model.

```
gauss% delphi 1 mm1 -endjob 10
```

The first required argument (a 1 in this case) specifies the first stream that is used in the simulation. In order to make the simulation technique of common random numbers as applicable as possible, random numbers for different purposes are drawn from as many distinct independent streams as is possible. In the output file **mm1.run**, Delphi displays exactly how many streams were used in the simulation. In our $M/M/1$ model, stream one is used for the arrival process, and stream two is used for the service time process. Thus to execute a second, independent run, we should use the command

```
gauss% delphi 3 mm1 -endjob 10
```

The second required argument specifies the input file name prefix. Delphi will attempt to read system configuration data from the file **mm1.del**. For systems with a single job type, the routing information is read from the file **mm1.r1**. If multiple job types are included in a model, the routing information for job type j is read from file **mm1.r**j.

Finally, in every simulation we must specify an ending condition. To specify an ending job number, we use the **-endjob number** option. Unless **-endjobtype type** is also specified, the simulation will terminate when **number** jobs of job type 1 have exited the system.

To specify an ending simulation clock time, use the **-endtime time** option. The simulation will terminate after the first event that occurs on or after **time**.

Below we list other commonly used options and a brief explanation of their purpose. Let **config** be the input file prefix (**mm1** in our example above).

All optional arguments to Delphi are not sensitive to case. The required file name, however, must be entered in the proper case.

-cilevel level Specifies the percent level of the confidence intervals given in the output statistics. Default is 95, for 95% confidence intervals.

-debug Writes an event by event debugging trace to **config.dbg**.

-debugev eventname Turns on debug trace, but only for event **eventname**. May be used in conjunction with other debug options.

-debugjob job Turns on debug trace, but only for job **job**. May be used in conjunction with other debug options.

-debugq queuename Turns on debug trace, but only for queue **queuename**. May be used in conjunction with other debug options.

-debugstart time Turns on debug trace, but only after simulation clock reaches **time**. May be used in conjunction with other debug options.

-endjob job Simulation ends when **job** jobs of job type 1 have exited system. To change the job type counted, use **-endjobtype**.

-endjobtype jobtype When used in conjunction with **-endjob**, specifies the job type to be counted for simulation ending condition.

-endtime time Simulation ends after the first event executed on or after **time**.

-jtnis Generate and report on number in system for each jobtype separately, as well as for all jobtypes in aggregate.

-nbatch batches Specifies the number of batches to use when calculating batch means and confidence intervals for time in system. Unless specified, Delphi uses an internal default. The number of batches used is displayed in **config.run**.

-nistrace Writes number in system trace for job type j to file **config.p**j.

-nohead Suppresses the heading normally printed in the debug trace output.

-norelease Suppresses regular job release mechanism. Jobs must be loaded into the system using the **-readstate** option.

-norunlist Simulation will not generate **config.run**. Used in conjunction with **-reps** when there are many replications, and output statistics are not needed for each individual replication.

-readstate file At the beginning of the first replication, Delphi reads in the contents of **file**, which should be a listing of jobs in the format generated by the **-savestate** option. When each replication is initialized, these jobs are released at the job step specified in **file**.

-repfiles Writes output files for different replications to separate files. Runlist output for replication k is written to **configr**k**.run**, debug output to **configr**k**.dbg**, and trace output for jobtype j to **configr**k**.p**j.

-reps replications Simulation is replicated **replications** times. Output files contain data from all replications. Use **-norunlist** to stop generation of **config.run** after each replication.

-saveint Specifies the interval between saves of state space information (location of all jobs).

-savestate If **-saveint** is not specified, at the end of each replication the location of all jobs is written to **config.s1**. If **-saveint** is specified, at the specified intervals the location of all jobs is written to **config.s**k, where k is incremented after each save operation.

-trunctime time Specifies the first time when statistics are collected. If **-endjob job** is specified, then the simulation will end when **job** jobs of the appropriate type have exited after **time**. Simulation stopping time is unaffected by **-trunctime** if **-endtime** is specified.

-xbtrace Writes time in system batch means trace for job type j to file **config.p**j.

-xtrace Writes job information trace for job type j to file **config.p**j.

19 RANDOM NUMBERS

Random numbers in Delphi are generated using the C version of a generator given in Chapter 7 of Law and Kelton (1991). Over 20,000 non-overlapping random number streams are provided, to enhance the applicability of common random numbers as a variance reduction technique (see Chapter 11 of Law and Kelton). Wherever possible within Delphi, separate streams are used. For example, each queue has separate streams for generation of service, failure, repair, and maintenance times. The number of streams used during a particular run is listed in the output file.

20 ESTIMATED NUMBER OF VISITS

In the output file **config.run**, Delphi displays for each job type the estimated number of times the job visits each queue before exiting the system. These estimates are only for those jobs that successfully exit the system, not for those that are scrapped along the way.

This section of output statistics is included so that a more complete picture of queuing bottlenecks can be achieved. For instance, suppose we estimate the delay at queue one to be ten hours, while at queue two to be thirty hours. However, if each job visits the first queue an average of ten times, while visiting the second tool only once, the total delay contributed by queue one is a much larger percentage of the overall time in system than that contributed by queue two. It might be more appropriate, then, to concentrate resources on lowering the delay at queue one.

21 PROGRAM VERIFICATION

The correctness of the program logic within Delphi can be verified in two ways. First, for sufficiently small and uncomplicated models, the limiting expected delay in system can be calculated analytically, and compared with simulation results for long runs. Second, a debugging trace can be generated, and compared with predicted behavior. We detail in this section a list of tests that can be easily performed to check several aspects of Delphi's logic.

21.1 M/M/S QUEUE

The M/M/s queue has exponentially distributed interarrival and service times, and s servers. Let λ be the arrival rate, μ the service rate, and $\rho = \lambda/(s\mu)$ the traffic intensity. For either first-come-first-served (also known as first-in-first-out, abbreviated FIFO), or last-come-first-served (LIFO) disciplines, the mean of the limiting time in system distribution is given by

$$\frac{u_s}{s\mu(1-\rho)^2},$$

where

$$u_s = \left(\frac{\lambda}{\mu}\right)^s \frac{1}{s!}\left[\sum_{i=0}^{s-1}\left(\left(\frac{\lambda}{\mu}\right)^i \frac{1}{i!} + \left(\frac{\lambda}{\mu}\right)^s \frac{1}{s!(1-\rho)}\right)\right]^{-1}.$$

When the service discipline is first-come-first-served, the variance of the limiting time in system distribution is given by

$$\frac{u_s}{(s\mu)^2(1-\rho)^4}(2-2\rho-u_s).$$

When the service discipline is last-come-first-served, the variance is given by

$$\frac{u_s}{(s\mu)^2(1-\rho)^4}(2-u_s).$$

In each of these equations, u_s is as given above.

These quantities are listed in Table 3 for several arrival and service rates, with varying number of servers and service discipline. The true mean and variance should be compared to the estimated mean and variance listed in the Delphi output file for runs of several hundred thousand customers.

Input files for the configurations listed in Table 3 are included in the Delphi distribution as **mmsA.del** and **mmsA.r1** (FIFO version), **mmsAL.del** and **mmsAL.r1** (LIFO version), **mmsB.del** and **mmsB.r1**, etc.

Number Servers s	Mean Arrival Rate λ	Mean Service Rate μ	Traffic Intensity ρ	Limiting Time in System Mean	Limiting Time in FIFO System Variance	Limiting Time in LIFO System Variance
1	0.5	2.0	0.25	0.667	0.444	0.519
2	2.5	1.53846	0.81	1.913	3.205	22.174
10	0.5	0.055	0.91	32.100	693.59	6260.97
25	0.2	0.01	0.80	104.182	10149.8	10818.9

Table 3: Verification table for M/M/s Queue

21.2 M/G/1 QUEUE

The M/G/1 queue has exponentially distributed interarrival times, general service times, and a single server. The service discipline is first-come-first-served. Let λ be the arrival rate, μ the service rate, and $\rho = \lambda/\mu$ the traffic intensity. From Equation (6.25) of Prabhu (1981), the mean of the limiting time in system distribution is given by

$$\frac{\lambda \mathrm{E}[S_0^2]}{2(1-\rho)} + \frac{1}{\mu},$$

or, if we substitute in the variance plus the squared mean for the second moment of the service time distribution,

$$\frac{\lambda\left(\mathrm{Var}[S_0] + \frac{1}{\mu^2}\right)}{2(1-\rho)} + \frac{1}{\mu}.$$

This limiting value is calculated in Table 4 for several service time distributions and values of λ and μ.

Input files for the configurations listed in Table 4 are included in the Delphi distribution as **mg1A.del** and **mg1A.r1**, **mg1B.del** and **mg1B.r1**, etc.

21.3 M/M/1 IN SERIES

Consider an assembly line of three machines, with each job having to visit the machines along the line exactly once, in sequence. If the interarrival time to the first machine or queue is exponentially distributed, and service times at all machines are exponentially distributed, then according to Section III.4 of Asmussen (1987), the limiting input process to all queues

Service Time Distribution	Mean Arrival Rate λ	Mean Service Rate μ	Traffic Intensity $\rho = \lambda/\mu$	Limiting Time in System
u(2,4)	0.10	1/3	0.300	3.67
u(5,10)	0.05	1/7.5	0.375	9.83
c(3)	0.10	1/3	0.300	3.64
c(7.5)	0.05	1/7.5	0.375	9.75
up(3,33.3)	0.10	1/3	0.300	3.67
tp(5,50)	0.067	1/5	0.333	6.30

Table 4: Verification table for M/G/1 Queue

is Poisson (interarrival times are exponentially distributed), and we may calculate limiting expected time in queue separately for each queue.

Suppose the arrival rate to the system is $\lambda = 1.0$, and the service rates at the three queues are $\mu_1 = 2.5$, $\mu_2 = 2.0$, and $\mu_3 = 2.5$. For simplicity, suppose each queue has exactly one server.

Thus, the total limiting expected time in system should be

$$\begin{aligned} \frac{1}{\mu_1 - \lambda} + \frac{1}{\mu_2 - \lambda} + \frac{1}{\mu_3 - \lambda} &= 0.67 + 1.0 + 0.67 \\ &= 2.33 \end{aligned}$$

The input files for this scenario are included with the Delphi distribution as **series.del** and **series.r1**.

21.4 M/M/1 IN SERIES WITH SCRAP

We test whole-job scrapping here only, so that we may compare with analytic results for M/M/1 queues in series.

Suppose we have three machines in series, with some scrap occurring after steps one and two. If the interarrival times to the first machine are exponentially distributed, and all service times are exponentially distributed, then we can analyze the limiting expected time in queue separately for each queue, as before. The only modification is that the arrival rate to each queue changes, as some jobs are removed due to scrapping.

Let the arrival rate to machine one be $\lambda_1 = 1.0$. If approximately one out of every ten jobs is scrapped during processing at machine one, then the arrival rate to machine two is

$\lambda_2 = \lambda_1 * 0.9 = 0.9$. If approximately one out of every five jobs is scrapped during processing at machine two, then the arrival rate to machine three is $\lambda_3 = \lambda_2 * 0.8 = 0.72$.

Suppose as before that the three processing rates are $\mu_1 = 2.5$, $\mu_2 = 2.0$, and $\mu_3 = 2.5$.

Then, the total limiting expected time in system should be

$$\begin{aligned} \frac{1}{\mu_1 - \lambda_1} + \frac{1}{\mu_2 - \lambda_2} + \frac{1}{\mu_3 - \lambda_3} &= 0.67 + 0.91 + 0.56 \\ &= 2.14 \end{aligned}$$

The input files for this scenario are included with the Delphi distribution as **scrap.del** and **scrap.r1**.

21.5 M/M/1 IN SERIES WITH REWORK

We test whole-job rework here only, so that we may compare with analytic results for M/M/1 queues in series.

Suppose we have two machines in series, with some possibility of rework at step one. By rework, we mean that there is a certain probability that for every job leaving step one, it may have to visit a third machine for processing, then pass through step one again.

Suppose all three machines have processing rate $\mu = 2.0$. Let the arrival rate into the system be $\lambda = 1.0$. However, that is not the effective arrival rate that machine one sees, due to the possibility of rework. Let the probability of rework be $p = 0.20$, that is, approximately one out of every five jobs will be reworked.

Jobs that are reworked visit a rework machine, then return to machine one. Jobs that are not reworked visit machine three, then exit the system.

To calculate the analytic limiting expected time in system, we need first to calculate the effective arrival rate to all three machines, given that rework can occur.

First, the arrival rate to machine one is incoming jobs at a rate of λ, plus reworked jobs at a rate of $p\lambda$, plus those jobs that are reworked twice at a rate of $p^2\lambda$, etc. Thus,

$$\begin{aligned} \lambda_1 &= \sum_{n=0}^{\infty} p^n \\ &= \frac{1}{1-p} \\ &= 1.25. \end{aligned}$$

The arrival rate to the rework machine is the probability of rework times the effective arrival rate into machine one,

$$\lambda_r = p\lambda_1$$

$$= 0.25.$$

The arrival rate to machine two is the exit rate of machine one (which must be its arrival rate λ_1) minus the rate of jobs being reworked (λ_r),

$$\begin{aligned} \lambda_2 &= \lambda_1 - \lambda_r \\ &= 1.0. \end{aligned}$$

Finally, the expected time in system is the expected number of visits to each queue times the expected time at queue, added together,

$$\begin{aligned} \frac{\lambda_1}{\lambda}\frac{1}{\mu-\lambda_1} + \frac{\lambda_r}{\lambda}\frac{1}{\mu-\lambda_r} + \frac{\lambda_2}{\lambda}\frac{1}{\mu-\lambda_2} &= 1.25(1.333) + 0.25 * (0.571) + 1.0 * (1.0) \\ &= 2.81. \end{aligned}$$

The input files for this scenario are included with the Delphi distribution as **rework.del** and **rework.r1**.

21.6 M/M/1 WITH TWO PRIORITY CLASSES

Consider a single server with two classes of jobs, one receiving high priority, the other receiving low priority. The discipline is head-of-the-line, in that high priority jobs that arrive during the service of a low priority job do not interrupt service. Rather, the high priority job moves ahead of any low priority jobs waiting for service. Within a priority class, service is first-come-first-served.

Let λ_H, λ_L denote the arrival rates for the two classes. Let μ_H, μ_L denote the service rates for the two classes. Denote the traffic intensities by $\rho_H = \lambda_H/\mu_H$, $\rho_L = \lambda_L/\mu_L$, and assume $\rho_H + \rho_L < 1$.

We obtain the limiting expected queuing delay from Table 4.4 of Prabhu (1981), and after adding the expected service time, we find the expected time in system for high priority jobs to be

$$\frac{\nu}{(1-\rho_H)} + \frac{1}{\mu_H},$$

and for the low priority jobs to be

$$\frac{\nu}{(1-\rho_H)(1-\rho_H-\rho_L)} + \frac{1}{\mu_L},$$

where

$$\nu = \frac{\lambda_L}{\mu_L^2} + \frac{\lambda_H}{\mu_H^2}.$$

Suppose $\lambda_H = 0.5$, $\lambda_L = 0.25$, $\mu_H = 2.0$, $\mu_L = 0.5$. Then $\nu = 1.125$, and the expected time in system for high priority jobs is

$$\frac{1.125}{0.75} + \frac{1}{2.0} = 2.0.$$

For low priority jobs, the expected time in system is

$$\frac{1.125}{(0.75)(0.25)} + \frac{1}{0.5} = 8.0.$$

The input files for this scenario are included with the Delphi distribution as **priority.del**, **priority.r1**, and **priority.r2**.

21.7 M/G/1 WITH LOADING AND UNLOADING

Consider a single server and single job class, but each service time consists of three parts: loading the job, servicing the job, and unloading the job. The service discipline is first-in-first-out.

We can use the formula given before for the $M/G/1$ queue to calculate the limiting expected time in system, where the service time has three parts,

$$\frac{\lambda \mathrm{E}[S_0^2]}{2(1-\rho)} + \frac{1}{\mu}.$$

Suppse $\lambda = 1.0$, and the three parts of the service time are distributed exponentially with means $1/\mu_1 = 0.1$, $1/\mu_2 = 0.4$, and $1/\mu_3 = 0.2$. Then

$$\begin{aligned} \frac{1}{\mu} &= \frac{1}{\mu_1} + \frac{1}{\mu_2} + \frac{1}{\mu_3} \\ &= 0.70. \end{aligned}$$

Now the second moment of the service time,

$$\begin{aligned} \mathrm{E}[S_0^2] &= \mathrm{Var}[S_0] + (\mathrm{E}[S_0])^2 \\ &= \frac{1}{\mu_1^2} + \frac{1}{\mu_2^2} + \frac{1}{\mu_3^2} + \frac{1}{\mu^2} \\ &= 0.70, \end{aligned}$$

since the service time is the sum of three independent exponential random variables. The traffic intensity $\rho = 0.70$. Hence the limiting expected time in system should be

$$\frac{1.0 * 0.70}{2 * (1 - 0.70)} + 0.70 \approx 1.87.$$

The input files for this scenario are included with the Delphi distribution as **load.del** and **load.r1**.

22 SUMMARY

Delphi is a queuing network simulator written in portable C code. It was developed to provide a platform for testing various scheduling and simulation algorithms on large, realistic queuing network models. Delphi is available via anonymous ftp from 'gauss.orie.cornell.edu'. There is no charge for Delphi if it is used for research at any degree granting institution. Before using Delphi in any commercial enterprise, a license must be obtained from the author.

ACKNOWLEDGMENTS

The author would like to thank Sarah Hood, Gerald Feigin, and Dan Friedman of IBM T. J. Watson Research Center for valuable comments and suggestions regarding semiconductor manufacturing simulation. Lee Schruben of Cornell University was quite helpful in discussions concerning appropriate statistical outputs.

REFERENCES

S. ASMUSSEN. 1987. *Applied Probability and Queues.* New York: John Wiley & Sons.

P. BRATLEY, B. FOX, AND L. SCHRAGE. 1987. *A Guide to Simulation.* New York: Springer-Verlag.

S. J. HOOD, A. E. B. AMAMOTO, AND A. T. VANDENBERGE. 1989. A modular structure for a highly detailed model of semiconductor manufacturing. In: *Proceedings of the 1989 Winter Simulation Conference*, eds. E. A. MacNair, K. J. Musselman, and P. Heidelberger, 811-817. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

L. LAMPORT. 1986. LaTeX: *A Document Preparation System.* Reading, Massachusetts: Addison-Wesley.

A. M. LAW AND W. D. KELTON. 1991. *Simulation Modeling and Analysis.* New York: McGraw-Hill.

N. PRABHU. 1981. Basic Queueing Theory. Technical Report No. 478, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, New York.

L. SCHRUBEN. 1991. *Sigma: A Graphical Simulation System.* San Francisco: Scientific Press.

L. SCHRUBEN, H. SINGH, AND L. TIERNEY. 1983. Optimal Tests for Initialization Bias in Simulation Output. *Operations Research* 31: 1167-1178.

P. WELCH. 1983. The Statistical Analysis of Simulation Results. *The Computer Performance Modeling Handbook.* New York: Academic Press.