Proceedings of the 2004 Winter Simulation Conference
R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

FROM EMERGENCY DEPARTMENTS TO COTTAGE CHEESE TO FIRE DEPARTMENTS: LEARNING SIMULATION EXPERIMENTATION THROUGH WEBGPSS

Richard G. Born

Department of Operations Management and Information Systems
College of Business
Northern Illinois University
DeKalb, IL 60115, U.S.A.

ABSTRACT

An important component of any simulation course is the discussion of experimental design. WebGPSS has been used by the author for two years to discuss experimentation in an introductory course in discrete-event simulation. This paper discusses how to set up simulation experiments using WebGPSS by presenting three business problems whose solutions require careful attention to experimental design. The first problem looks at staffing an emergency department of a hospital with physicians. The second problem involves the optimal method for stocking a perishable food product such as cottage cheese on a supermarket shelf, showing that under realistic conditions, spoilage can be minimized by placing newer containers in the front. The third problem involves analysis of a proposal for two neighboring communities currently operating completely separate fire departments to integrate their two systems in a way that could reduce the amount of time that fires are unattended.

1 INTRODUCTION

WebGPSS (Ståhl 2003) is the most modern implementation of micro-GPSS, a streamlined version of GPSS, the General Purpose Simulation System, which originated more than 40 years ago. WebGPSS has been in existence for about three years and has been used by students in Sweden (Herper and Ståhl 2003) and in the United States (Born 2003). In addition, I. Ståhl, developer of WebGPSS, has taught simulation modeling using WebGPSS to English-speaking students throughout Europe.

A variety of features of WebGPSS make it particularly appropriate for teaching simulation modeling to business students. First, it has a Graphical User Interface that allows students to build models graphically in the form of block diagrams, supplying operands by double-clicking on the blocks within the block diagram. Second, it is available both on the Web and as a stand-alone version on a CD. Third, it has many pedagogical simplifications, including fewer block symbols, making it much easier to learn than traditional GPSS. Fourth, it has an extensive error trapping and reporting mechanism with more than 500 error codes, accomplished with the help of several thousand GPSS students over the years who have been asked to report any errors for which the error code is not understandable or helpful. Fifth, a multi-language focus has made it relatively straightforward to develop WebGPSS systems in Swedish, and English, and there is now some interest in the development of both French and Spanish versions. Sixth, WebGPSS has a complete teachware package (Born and Ståhl 2003) containing more that 500 PowerPoint slides designed for learning and teaching all of the material covered in Ståhl (2003). Some of the features of this teachware package can be found in Schriber *et al*. (2003). Last, and of particular interest in this paper, is the ability to set up and perform simulation experiments, with provision for both text output in the form of confidence intervals, and a histogram of result variable values.

We begin by discussing how to set up simulation experiments using WebGPSS. This is accomplished by looking at a very simple problem involving staffing the emergency department (ED) of a hospital with physicians. We then study a problem that we shall refer to as the *cottage cheese problem*. Here we will find a rather surprising result that under realistic conditions, spoilage of cottage cheese can be minimized by placing the newer containers in the front of a supermarket shelf. Finally, we will investigate a proposal to integrate two neighboring community fire departments in a way that could potentially reduce the amount of time that fires are unattended.

2 STAFFING THE EMERGENCY DEPARTMENT OF A HOSPITAL WITH PHYSICIANS

One of the most difficult problems facing hospital emergency departments worldwide is the reduction of patient

waiting times. Mahapatra *et al*. (2003) have developed a simulation model for the entire care delivery system that exists at an academic emergency department in York Hospital, Pennsylvania. This model is quite comprehensive and involves a sequence of activities including arrival, triage, registered nurse (RN) assessment, MD assessment, initial diagnosis and treatment, diagnostic testing, junior doctor supervision/teaching, follow up/treatment planning, discharge or admit, and access to inpatient beds and admitting physicians. In order to introduce simulation experimentation using WebGPSS, we shall consider an extremely small, simplified portion of what is involved in modeling a hospital emergency department—staffing the ED with physicians.

2.1 Problem Statement

Patients arrive at the ED on average every 6 minutes, exponentially distributed. Each patient requires an average of 45 minutes of a physician's time, and this time is normally distributed with a standard deviation of 8 minutes. For simplicity it can be assumed that the physician visits the patient just once, and does not return to that patient later. We want to investigate how the time a patient waits to be examined by a physician is related to the size of the pool of physicians. Since an ED is operational 24/7, we will run our simulation model for a period of 24 hours.

2.2 The WebGPSS Model

The WebGPSS block diagram for our simplified ED is shown in Figure 1. It consists of two segments, the patient segment on the left and the stop segment on the right.

We first discuss the patient segment. The GENERATE block produces the exponentially distributed patient arrivals, using the built-in exponential distribution function and random stream 1. The ADVANCE block produces the normally distributed physician service times, using the built-in standard normal function and random stream 2. Using a separate random number stream for each of the two random input variables in our model ensures that the inter-arrival and service time are the same for each successive patient as the experimental variable changes. The ENTER and LEAVE blocks are used, respectively, to access a physician from the storage EDPHYS and then free the physician when the service is completed. The LET blocks on either side of ENTER are used to compute the time that the patient waits, with this time stored in a parameter called P$WAIT. In the LET block preceding ADVANCE, these wait times are accumulated in a savevalue called X$TOTWT. This final let block has an address COUNT. N$COUNT, referenced in the stop segment, then represents the total number of patients who have entered this let block, i.e. that total number of patients who have finished waiting.

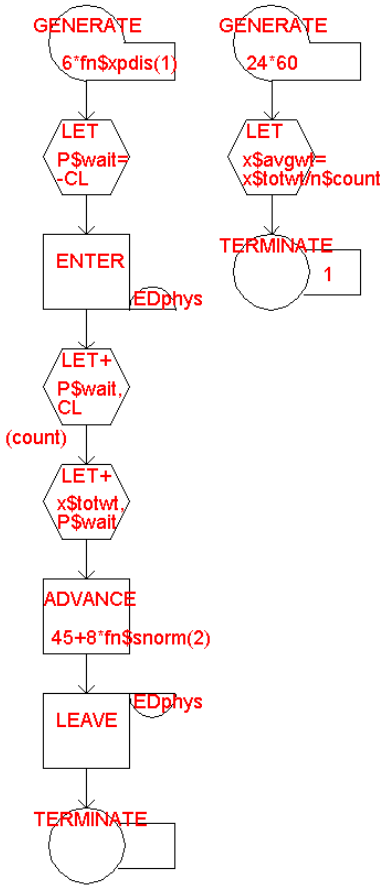


Figure 1: The Emergency Department Block Diagram

Regarding the stop segment, the simulation is run for 24 hours of 60 minutes each, as indicated by the GENERATE block. The LET block computes the average waiting time X$AVGWT, and the TERMINATE block brings the simulation to a stop.

The number of physicians available in the ED is defined in the Capacities window as shown in Figure 2. We see that WebGPSS allows defining the capacity of a storage by the use of a savevalue, which we call X$NUMPHY.

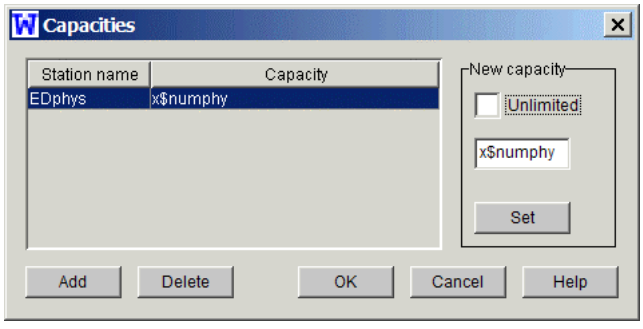
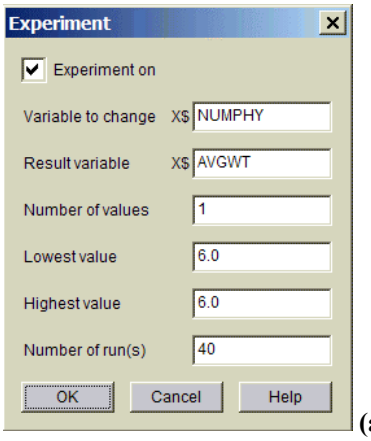


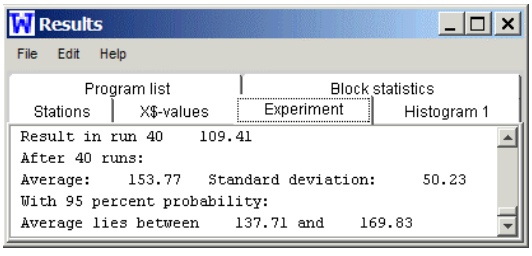
Figure 2: The Capacities Window

2.3 Three Experiments with the Emergency Department Model

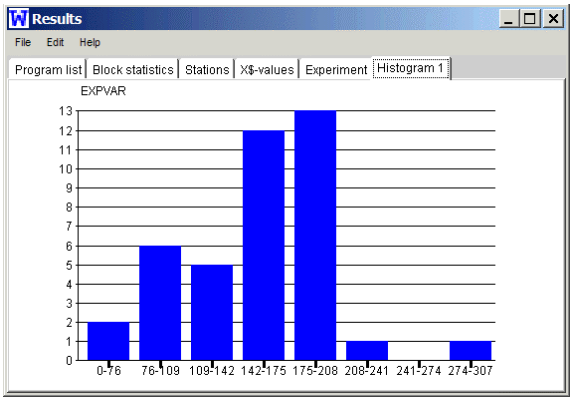
We first run an experiment 40 times with six physicians. Doing this would allow us to compare the experimental results for patient waiting time to actual ED waiting times, providing some validation for our WebGPSS model. Figure 3 summarizes the results of this experiment.



(a)



(b)

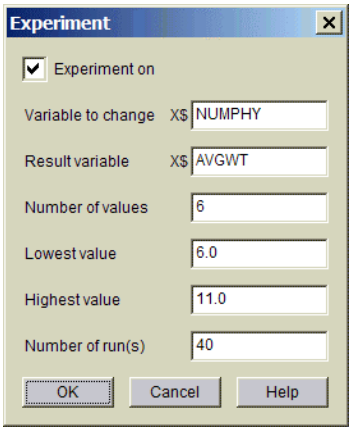


(c)

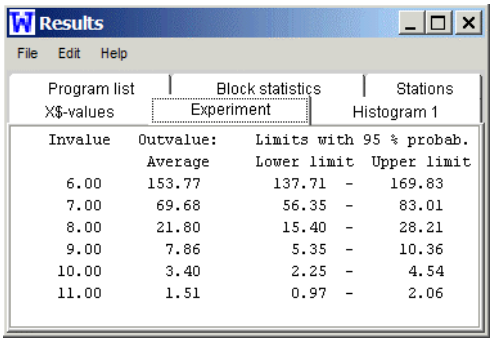
Figure 3: ED Experiment with 6 Physicians

Figure 3(a) shows the Experiment window, with X$NUMPHY as the experimental *variable to change* and X$AVGWT as the *result variable*. Figure 3(b) shows the output results after 40 runs, indicating a 95% confidence interval for average waiting time between 137.71 and 169.83 minutes. Figure 3(c) shows the WebGPSS histogram of average patient waiting times (i.e. EXPVAR, the experiment variable) for the 40 runs. We see that, for example, in 6 runs, the average waiting time was between 76 and 109 minutes. Clearly, six physicians are by no means adequate in number to meet patient needs.

In our second 40-run experiment with the ED model, we will let the number of physicians available for service vary from 6 to 11. Figure 4(a) shows the Experiment window, and Figure 4(b) shows the output results. *Invalue* refers to the values for the experimental variable X$NUMPHY, *Output Average* refers to the average value for the result variable X$AVGWT, and the two columns to the far right of the experiment results window provide 95% confidence intervals for the mean value of the patient waiting time. While we would naturally expect the waiting time to decrease as the number of physicians increases, this type of experiment can also be used for optimization in models where either a minimum or maximum value is desired for a result variable.



(a)

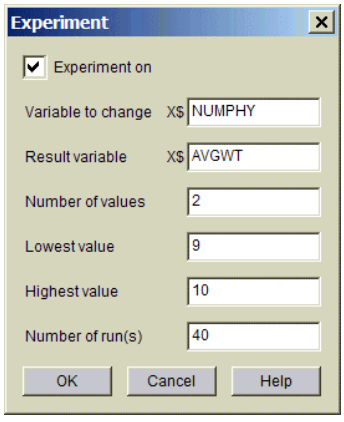


(b)

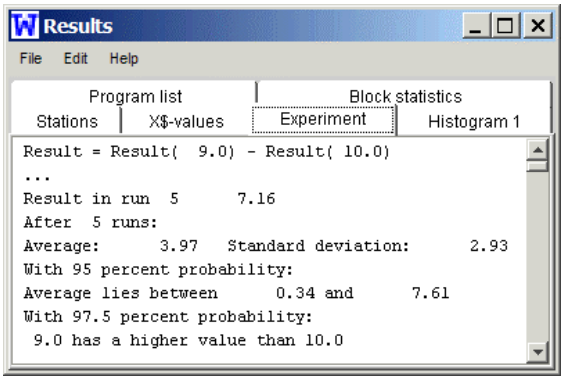
Figure 4: ED Experiment with the Number of Physicians Varying from 6 to 11

In our final ED experiment, we will show how WebGPSS can be used to perform pair-wise comparisons. Figure 5(a) shows the Experiment window. Whenever the *Number of values* is set to 2, WebGPSS will always perform a pair-wise comparison experiment, with the result being the difference *Result(Lowest value)-Result(Highest value)*. Figure 5(b) shows that the output result is the difference *Result(9 physicians)-Result(10 physicians)*. We see that after only five runs, the 95% confidence interval is

entirely positive, indicating that the average waiting time is greater for 9 physicians than it is for 10 physicians. The greatest advantage of doing pair-wise comparisons is that one can reach valid conclusions after a smaller number of simulation runs. We shall see in the remaining sections of this paper that pair-wise comparisons are also extremely useful when comparing the results of two completely different scenarios.



(a)



(b)

Figure 5: ED Experiment Using Pair-Wise Comparison

3 THE COTTAGE CHEESE PROBLEM

First, we want to provide some motivation for this problem. The author has noticed the following regarding the placement of milk gallons when shopping in the supermarket. There are several rows of milk gallons, some more at eye level, and others at a lower level requiring one to bend down to grab the gallon. To the author's surprise, the dates on the milk gallons on the eye level shelves are the new dates, while those at the bottom level shelves are the older dates. Intuition, perhaps misled, had suggested to the author that the store manager would want to place the older gallons at eye level, so that being easier to reach, customers would be more likely to grab them first. This scenario then led the author to devise a slightly modified problem involving cottage cheese. This is an extremely interesting problem that should be studied by all involved with inventory theory.

3.1 Problem Statement

The manager of a local supermarket has a single, deep shelf for cottage cheese. A customer who comes to buy a container of cottage cheese will always grab the one at the front of this deep shelf. In order to avoid spoilage, and hence monetary loss or loss of reputation to the supermarket, how should the stock boy be instructed to place newly arriving containers of cottage cheese? Should he place them at the front of the shelf so that customers will grab them first, or should he place them at the back of the shelf, behind containers that have not yet been sold? Assume the following parameters regarding this problem:

- The shelf can hold 20 containers of cottage cheese.
- Every 3.5 days (Monday morning and Thursday afternoon) new cottage cheese containers arrive from the distributor. The distributor leaves just enough containers to fill the shelf (e.g., if there were five containers on the shelf when the distributor arrived, she would leave 15 new containers).
- Customer arrivals are exponentially distributed with an average of one day apart.
- A container is considered spoiled if it has been on the shelf for more than 10 days when it is sold.
- You simulate one year (365 days) and provide for statistics on the number of containers that were sold in a spoiled state for the two different methods for stocking the shelf.

3.2 The WebGPSS Model

To conserve space in this paper, rather than provide the block diagram view of the model, we show the WebGPSS program listing in Figure 6. While comments on individual lines of code provide details regarding the logic involved in the model, a few general remarks are in order.

In the program listing, we first see two HELP control statements. WebGPSS provides a HELP control statement that allows the interactive input of values for savevalues. The user is asked to key in a value of 1 for X$POLICY if containers are to be placed on the shelf in FIFO fashion, and a value of 2 if LIFO is used. The window that the user sees for inputting the value of X$POLICY is shown in Figure 7. Similarly, the user interactively inputs the number of cottage cheese containers, X$SHELF, that fit on the shelf.

The program consists of three segments. The first segment simply sets the simulation to run for one year (365 days). The second segment creates exponential customer arrivals with an average of one day apart, giving these arrivals higher priority (999999) than transactions created in

the other segments. W$WAITIN refers to the current contents of the block whose address in WAITIN.

```
        simulate  1

 ! FIFO (1) or LIFO (2)?
        HELP      INPUT,X$policy
 ! How many cottage cheese cups will fit on the shelf?
        HELP      INPUT,X$shelf
        QTABLE    time,0,10,2
        GENERATE  365         ! Simulate for one year
        TERMINATE 1

        GENERATE  fn$xpdis,,,,999999 ! Customer arrivals
        IF        w$waitin=0,empty ! Go home if no cups on shelf
        SEIZE     signal      ! Let order seg. know customer is here
        ADVANCE   0.0010      ! Give order seg. chance to do its job
        RELEASE   signal      ! Let order seg. know customer is done
        TERMINATE
 empty  TERMINATE            ! Will tally times when shelf is empty

        GENERATE  3.5,,0      ! Distributor arrives every 3.5 days
        LET       x$nucups=x$shelf-(w$joint+w$waitin)
        IF        x$nucups=0,bye ! Leave if shelf is full
        IF        x$policy=1,fifo ! Allow user input of FIFO/LIFO
        LET       PR=CL ! Will set up policy as LIFO
        GOTO      makcop
 fifo   LET       PR=-CL ! Will set up policy as FIFO
 makcop LET       x$copies=x$nucups-1 ! - 1 for the "original" cup
        IF        x$copies=0,joint
        SPLIT     x$copies,joint ! Fill the shelf with cups
 joint  ARRIVE    time        ! Start of shelf time
 waitin SEIZE     shelf       ! Get position in front of shelf
        WAITIF    signal=NU   ! Wait if a customer is not in store
        WAITIF    signal=U    ! Wait until customer ready to leave
        RELEASE   shelf       ! Free shelf for next available cup
        DEPART    time        ! End of shelf time
 bye    TERMINATE

        start     1
        end
```

Figure 6: WebGPSS Program Listing of the Cottage Cheese Model

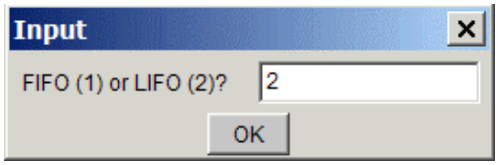


Figure 7: Input Window for Shelf Stocking Policy, with LIFO Selected

The third segment in our cottage cheese model is the most complex:

- It generates cottage cheese distributor arrivals every 3.5 days.
- X$COPIES is one less than the number of containers of cottage cheese that the distributor must leave to fill the shelf.
- WebGPSS priority can be real numbers—negative, zero, or positive, with larger numbers representing higher priority. Therefore, assigning the simulation clock's value (CL) to a transaction's priority will ensure a LIFO policy, in which newly arriving containers get placed in the front of the shelf. On the other hand, assigning the negative of the simulation clock's value (-CL) ensures a FIFO policy, with newly arriving containers being placed in the back of the shelf.
- The WebGPSS SPLIT block makes copies of transactions that inherit PRIORITY from the original. The SPLIT block in our model creates enough containers to fill the shelf with cottage cheese containers, when both the copies and the original are considered.
- The ARRIVE block starts measuring the time a container spends on the shelf before being purchased by a customer. The DEPART block concludes measurement of this shelf time.
- Only one container at most can be at a position in the front of the shelf at any given time. The SEIZE and RELEASE blocks accomplish this in our model.
- The WebGPSS WAITIF block causes transactions to wait if and as long as a condition is true, with waiting transactions actually waiting in the block immediately preceding the WAITIF. The container in the front of the shelf must first wait if and as long as a customer is not in the store (SIGNAL=NU, for *not in use*). Then it must wait while a customer is actually picking the container from the front of the shelf (SIGNAL=U, for *in use*).

3.3 Output Analysis

We note that there is a QTABLE control statement in the listing of the cottage cheese model shown in Figure 6. This queue table computes statistics for the ARRIVE/DEPART set TIME, which represents the shelf time for cottage cheese containers. The upper limit for the lowest class in 0 minutes, the width of each class is 10 minutes, and there are 2 classes. Therefore, any times greater than 10 minutes will be tallied as an overflow, and would represent cottage cheese containers that were sold in a spoiled state.

Figure 8 shows the queue table for FIFO stocking of a shelf that can hold 20 cottage cheese containers, while Figure 9 shows the corresponding results for LIFO stocking. A number of observations can be made when comparing these two stocking policies:

- The mean shelf time with the LIFO policy (8.11 days) is less than half the mean shelf time for the FIFO policy (17.98 days).
- The percentage of cottage cheese containers sold in a spoiled state for the LIFO policy is only 17.60%, compared to 96.37% for the FIFO policy.
- The variability of shelf time is much greater for the LIFO policy. This is shown by comparing the standard deviation in shelf time for the two policies, the maximum time, and the average value of

the overflow. Apparently, some cottage cheese containers under a LIFO policy spend a lot of time near the back of the shelf.

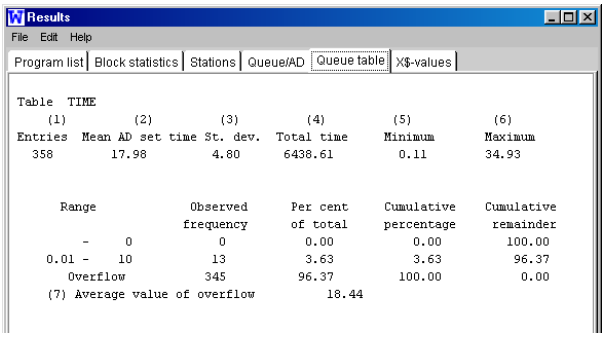


Figure 8: Shelf Times with FIFO Stocking

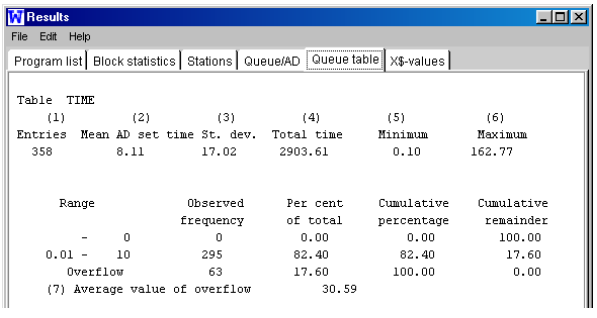


Figure 9: Shelf Times with LIFO Stocking

Clearly, under realistic conditions, it makes sense to use a LIFO policy and have the stock boy place newly arriving cottage cheese containers at the front of the shelf, thus avoiding significant spoilage. Having studied this problem, the author now understands why the supermarket discussed at the beginning of this section placed newly arriving milk gallons at eye level and older gallons at a more difficult to reach lower level.

Although this problem consists of a relatively small number of blocks, 26, the logic and use of WebGPSS constructs are complex enough that one can expect that a relatively small number of students would be able to come up with a solution on their own. However, once presented with the solution, students can readily make modifications to the model that would compare the two stocking policies using the WebGPSS Experiment window along with some of the techniques discussed in Section 2 of this paper.

4 THE FIRE DEPARTMENTS PROBLEM

We finally discuss a somewhat modified version of a simulation application problem originally appearing in Watson and Blackstone (1989). This problem involves the analysis of a proposal to integrate the fire departments from two neighboring communities if it can be shown that doing so would reduce the amount of time that fires are unattended. The original problem in Watson (1989) was designed as a pencil-and-paper problem for students to analyze, given historical wall clock times and corresponding service times for fire calls for each of the two communities. The author of this paper viewed this as an ideal problem for WebGPSS scenario analysis. We begin by looking at a detailed statement of the problem.

4.1 Problem Statement

Two neighboring communities, Springdale and Winterville, currently operate completely separate fire departments. A proposal has been made, however, to the city councils of the two communities, to integrate the two fire departments. Anytime that a fire truck is not available in one of the communities, a "hot line" call would be made to the fire station in the other community. If a fire truck is available there, it would answer the call. If not, the call would revert back to the original community to be handled by a fire truck there as soon as one is available. Due to political considerations, the proposal to integrate the two fire departments will be seriously considered only if it can be shown that a merger will reduce the amount of time that fires are unattended. The following additional facts and assumptions apply:

- Any fire call requires service from exactly one truck.
- Springdale has two fire trucks and Winterville has one.
- Fire call arrivals for Springdale are exponentially distributed with a mean of 4 hours, while fire call arrivals for Winterville are exponentially distributed with a mean of 5 hours.
- Service times for all calls, regardless of the community of origin, are randomly distributed as follows:
  - 30% of the calls require 0.5 hour
  - 40% of the calls require 1 hours
  - 20% of the calls require 1.5 hours
  - 10% of the calls require 2 hours
- To answer a call in the other community adds ½ hour to the service time. This time is evenly divided between ¼ hour to drive to the call and ¼ hour to return to the station.
- Assume that fires responded to by a community's own fire truck are instantaneous, i.e. the time for the truck to drive to the fire is negligible.

Design a WebGPSS experiment that will provide the city councils with information on whether or not the merger would reduce the amount of time that fires are unattended. Run the model for one year of system operation.

4.2 The WebGPSS Model

The WebGPSS program listing for the fire departments model is shown in Figure 10. As in the previous model, comments on individual lines of code provide details regarding the logic involved in the model, but we will provide some discussion regarding how this model was built using the WebGPSS graphical user interface.

```
        simulate  1
winser FUNCTION  RN2,R
0.5 30
1 40
1.5 20
2 10
sprser FUNCTION  RN4,R
0.5 30
1 40
1.5 20
2 10
! Scenario (1=separate, 2=combined)?
       HELP      INPUT,X$SCENAR
sprtrk CAPACITY  2
wintrk CAPACITY  1
       QTABLE    time,0,1,20

       GENERATE  4*fn$xpdis(1) ! Fire call at Springdale
       ARRIVE    time      ! Start measuring unattended time
       LET       P$TIME=-CL  ! Parameter with start time
       IF        X$SCENAR=1,sprblk ! Separate if 1, combined if 2
       IF        sprtrk=F,trywin!Springdale trucks busy-try W'vill
sprblk ENTER     sprtrk      ! Get a Springdale truck
       DEPART    time        ! End measuring unattended time
       LET+      P$TIME,CL   ! Parameter now has unattended time
BL1    LET+      X$TOTTIM,P$TIME ! Accumulate unattended times
       ADVANCE   fn$winser   ! Put out the Springdale fire
       LEAVE     sprtrk      ! Free the Springdale truck
       TERMINATE             ! End of this fire call incident
trywin IF        wintrk=F,sprblk ! W'ville busy, back to S'dale
       ENTER     wintrk      !Get a Winterville truck
       ADVANCE   0.25        ! Drive truck to Springdale
       DEPART    time        ! End measuring unattended time
       LET+      P$TIME,CL   ! Parameter now has unattended time
BL2    LET+      X$TOTTIM,P$TIME ! Accumulate unattended times
       ADVANCE   fn$sprser   ! Put out the Springdale fire
       ADVANCE   0.25        ! Drive truck back to Winterville
       LEAVE     wintrk      ! Free the Winterville truck
       TERMINATE             ! End of this fire call incident

       GENERATE  5*fn$xpdis(3) ! Similar code for Winterville
       ARRIVE    time
       LET       P$TIME=-CL
       IF        x$SCENAR=1,winblk
       IF        wintrk=F,tryspr
winblk ENTER     wintrk
       DEPART    time
       LET+      P$TIME,CL
BL3    LET+      X$TOTTIM,P$TIME
       ADVANCE   fn$sprser
       LEAVE     wintrk
       TERMINATE
tryspr IF        sprtrk=F,winblk
       ENTER     sprtrk
       ADVANCE   0.25
       DEPART    time
       LET+      P$TIME,CL
BL4    LET+      X$TOTTIM,P$TIME
       ADVANCE   fn$winser
       ADVANCE   0.25
       LEAVE     sprtrk
       TERMINATE
       GENERATE  24*7*52     ! Run for a year (time in hours)
       LET       X$AVGHRS=X$TOTTIM/(N$BL1+N$BL2+N$BL3+N$BL4)
       LET       X$AVGTIM=X$AVGHRS*60 ! Minutes = more precision
       TERMINATE 1           ! Stop the simulation

       start     1
       end
```

Figure 10: Program Listing of the Fire Department Model

First, we note that there are two FUNCTION control statements. These were created for the purpose of providing the random service times for fire calls. The Random Function window that was used to create the function called WINSER (for Winterville service times) is shown in Figure 11.

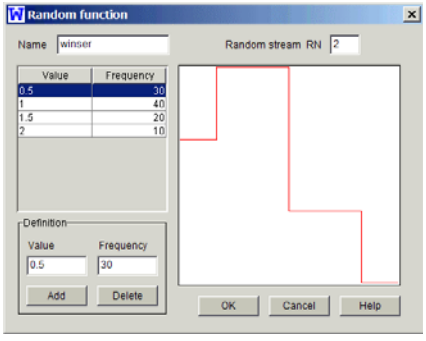


Figure 11: The WebGPSS Random Function Window

The user specifies the random stream to be used (stream 2) and keys the values and frequencies in the Definition frame. The corresponding graph is shown to the right in the Random function window. A similar function SPRSER (for Springdale service times) uses a different random number stream (stream 4), so that service times are controlled for each city from one scenario to another.

We next see a HELP control statement that allows interactive input for the scenario. If X$SCENAR has the value 1, then we are modeling the separate fire departments, else we model the proposed combined fire departments. Figure 12 shows the Start values window in which the model developer sets up the capability to interactively input the value of X$SCENAR via the prompt *Scenario (1=separate, 2=combined)?*

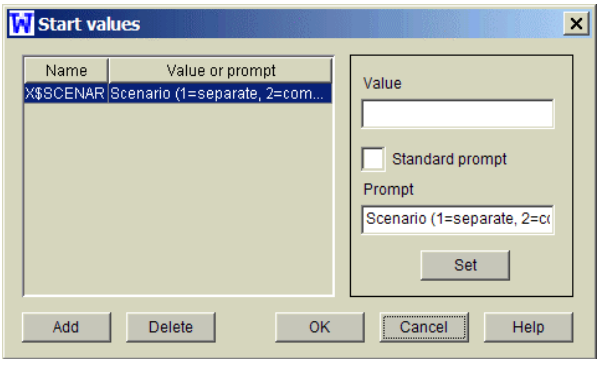


Figure 12: The WebGPSS Start Values Window with a User Defined Prompt

The CAPACITY control statements shown in Figure 6 are used to define the number of fire trucks for each of the two communities. As an example, Figure 13 shows how the capacity for Springdale was set to 2 fire trucks via the Capacities window.

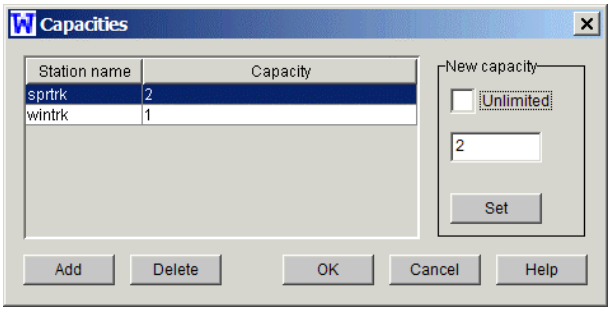


Figure 13: The WebGPSS Capacities Window

4.3 Output Analysis

We note that there is a QTABLE control statement in the listing shown in Figure 10. This queue table computes statistics for the ARRIVE/DEPART set TIME, which represents the time during which a fire is unattended after a call comes in to a fire department. The upper limit for the lowest class in 0, the width of each class is 1 hour, and there are 20 classes.

Figure 14 shows the queue table results for the scenario where the two community's fire departments remain separate, while Figure 15 shows the queue table results for the scenario for their proposed merger. At least two observations can be made when comparing the two scenarios:

- The average unattended time for fires with separate fire departments (0.08 hour) is more than twice that with the fire departments combined according to the proposal (0.03 hour).
- With separate departments, 2.83% of the fires are unattended for more than one hour. With combined fire departments, this figure is ten times smaller, at only 0.28%.

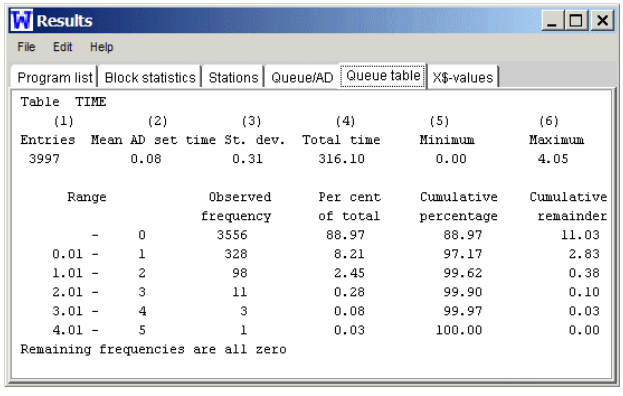


Figure 14: Queue Table of Unattended Times for Fires under the Scenario of Separate Fire Departments

Let us now see how to make use of the WebGPSS experiment window to compare unattended times for fires under the two scenarios. Here, we design a pair-wise comparison experiment in which we look at the *difference* in unattended times for fires under the two scenarios. First, it is noted from Figure 10 that X$TOTTIM has been accumulating the unattended times for fires. In the stop segment, the first LET block computes the average unattended time in hours, X$AVGHRS, by dividing the accumulated total by the number of incidents. X$AVGTIM then converts this to minutes for more precision. We can, therefore, let X$SCENAR be our experimental variable, and X$AVGTIM be our result variable, with an experiment designed as shown in Figure 16. The Results window of Figure 17 shows that after only three runs, we can conclude with at least 97.5 percent probability that the average unattended time for fires is about 3 minutes or 0.05 hours greater if we continue to run the two communities fire departments separately. This, we also note, agrees nicely with the results obtained from the queue table statistics of Figures 14 and 15, where we see the difference would be 0.08 – 0.03 = 0.05 hour. It appears that the city councils of the two communities have ample evidence that a merger of their fire departments should reduce the amount of time that fires are unattended.

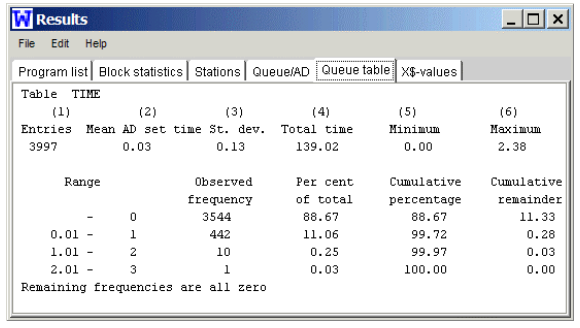


Figure 15: Queue Table of Unattended Times for Fires under the Scenario of Combined Fire Departments

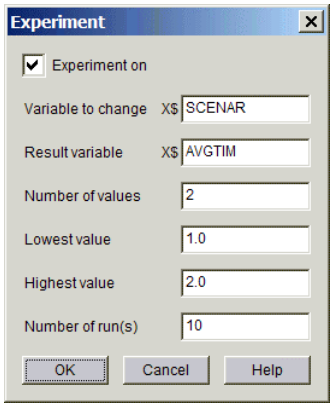


Figure 16: Pair-Wise Comparison Experiment Comparing Two Fire Department Scenarios. The Experiment Window

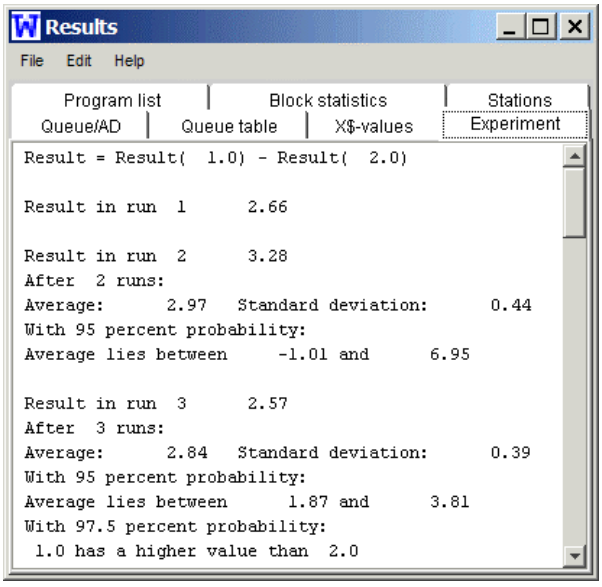


Figure 17: Pair-Wise Comparison Experiment Comparing Two Fire Department Scenarios. The Results Window

5 CONCLUSIONS

We have, in this paper, presented a primer on how to set up and design experiments using WebGPSS. The three models discussed—the hospital emergency department problem, the cottage cheese problem, and the fire departments problem—provide the reader with a variety of business problems having a wide range in complexity. These problems show that one can develop very realistic and useful simulation models with relative ease using WebGPSS. All of this model building is accomplished using a graphical user interface that is both intuitive and powerful. Any reader of this paper who is interested in obtaining copies of any or all of the models discussed, ready for opening in WebGPSS, is encouraged to contact the author at the email address provided in the biography.

REFERENCES

Born, R. 2003. Teaching discrete-event simulation to business students: the alpha and omega. In *Proceedings of the 2003 Winter Simulation Conference*, ed. S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, 1964-1972. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Born, R. and I Ståhl. 2003. *WebGPSS Slide Presentation*. DeKalb, IL: R. Born. Available on request from R. Born at <rborn@niu.edu>.

Herper, H. and I. Ståhl. 2003. Modeling and simulation in high schools – two European examples. In *Proceedings of the 2003 Winter Simulation Conference*, ed. S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, 1973-1981. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Mahapatra, S., C. P. Koelling, L. Patvivatsiri, B. Fraticelli, D. Eitel, and L. Grove. 2003. Pairing emergency severity index5-level triage data with computer aided system design to improve emergency department access and throughput. In *Proceedings of the 2003 Winter Simulation Conference*, ed. S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, 1917-1925. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Schriber, T., I. Ståhl, J. Banks, A. Law, A. Seila, and R. Born. 2003. Simulation textbooks – old and new (panel). In *Proceedings of the 2003 Winter Simulation Conference*, ed. S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, 1952-1963. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Ståhl, I. 2003. *Simulation made simple with WebGPSS – a tutorial*. Stockholm: Stockholm School of Economics.

Watson, H. and J. Blackstone, Jr. 1989. *Computer simulation – 2nd edition*. New York: Wiley.

AUTHOR BIOGRAPHY

RICHARD G. BORN is an Associate Professor of Management Information Systems in the Department of Operations Management and Information Systems in the College of Business at Northern Illinois University. He has taught simulation modeling for the past 13 years to university students at all levels from undergraduate to graduate, including M.S. students in Management Information Systems, M.S. students in Accountancy, and M.B.A. students. His email address is <rborn@niu.edu>.