USE OF AN EXTERNAL OPTIMIZING ALGORITHM WITH A GPSS MODEL

ROBERT M. LEFKOWITS
ARTHUR ANDERSEN AND COMPANY
CHICAGO, ILLINOIS

THOMAS J. SCHRIBER
GRADUATE SCHOOL OF BUSINESS
THE UNIVERSITY OF MICHIGAN
ANN ARBOR, MICHIGAN

ABSTRACT

Although GPSS enjoys many advantages as a language for modeling discrete, non-deterministic systems, it suffers from several disadvantages. GPSS is interpretive, resulting in relatively slow execution times. In addition, the input/output and computational capabilities in the language are limited. Disadvantages like these can be lessened or overcome by use of the GPSS HELP Block, which makes it possible to interface an executing GPSS model with one or more FORTRAN subroutines. As an instructive example of HELP Block utility, this paper shows how a GPSS model has been mated with a FORTRAN subroutine implementing the univariate searching strategy. The result is a GPSS-FORTRAN combination which automates the search for the optimal way to configure a system.

1. Introduction

A frequent use of simulation is to compare alternative system configurations with the purpose of finding the one which best satisfies a given objective. This comparison of alternatives often takes place on a man-model interactive basis. The analyst configures the model in some way, simulates with that configuration, examines the resulting measure of performance, estimates some other system configuration which might improve the performance measure, re-configures the model accordingly, and so on. Such an interactive process can be very time consuming. There is some motivation, then, to attempt automating the contribution of the man in this man-model interaction, so the need for interaction can be eliminated.

In many circumstances, the man's contribution cannot be automated. This is because it may be difficult or impossible to develop an algorithm which duplicates a man's heuristic decision-making ability. In other circumstances, however, the searching strategy a man might employ can be duplicated, or at least approximated to a satisfactory degree. If this is done, and if the resulting algorithm is suitably combined with a model simulating the system to be optimized, the search for optimality can be conducted automatically and quickly.

Whether a searching strategy can be automated depends on the behavior of the measure of performance, i.e., on the nature of the objective function. If the objective function is unimodal with respect to each of the independent variables, a variety of searching strategies is available [1,4]. A function is unimodal "if the independent variables are altered in an (arbitrarily) given direction, and if the resulting values of the function are found initially to increase up to a maximum and then to fall away again. In such functions, as the name suggests, there is only a single peak." [1, p. 108] The concept of unimodality is further motivated with an example in Section 3 below. Assuming unimodality, the univariate searching strategy can be implemented to seek the conditions under which the objective function assumes its maximum (or minimum) value. This strategy involves moving through the search domain along successive contours until all more promising directions of movement have been exhausted. Section 3 elaborates further on the method.

In this paper, a fully developed example is given to show how the FORTRAN-implemented univariate searching strategy can be combined with the GPSS model of a system, thereby automating the optimization process. A simple systems problem is first stated. The possibility of using the univariate searching strategy with a model of the system is then explained. A FORTRAN subroutine implementing this strategy is presented, and a GPSS model which simulates the system, and arranges for the necessary linkages with the FORTRAN routine, is discussed. Results produced by using the GPSS-FORTRAN combination are then presented.

2. Statement of the Problem

The manufacture of a certain line of widgets involves a relatively lengthy assembly process, followed by a short firing time in an oven. Since ovens are expensive to maintain, assemblers share a group of ovens. Each oven holds only one widget at a time. An assembler cannot begin assembling a next widget until he has removed his current one from the oven. This is the pattern followed, then, by each assembler.
(1) Assemble the next widget.

(2) Wait, first-come, first-served, to use the oven.
(3) Use the oven.
(4) Return to (1).

Time and cost studies have been conducted to provide the information in Tables 1 and 2, respectively.

Operation	Time Required, Minutes [a]
Assemble	30±5
Fire	8±2

Table 1. Operation Time Information

Item	Cost Information
Assembler's Salary	$3.75 per hour
Oven Cost	$80 per 8-hour work day (independent of utilization)
Raw Material	$2 per widget
Value of Finished Widgets	$7 per widget

Table 2. Financial Data

Marketing studies show that demand for widgets will support an average manufacturing rate of 275 widgets per working day (8-hour day).

Build a GPSS model for this manufacturing process. Then use the model to determine the optimal system configuration, i.e., the number of assemblers to hire, and ovens to purchase, to maximize profit. Base the determination on simulations equivalent to 10 8-hour working days. Assume there are no discontinuities within a working day, or in moving between consecutive 8-hour work days. Before making a given 10-day run, simulate with the configuration for 1.5 days to eliminate the effect of transient conditions in the system [b].

3. The Univariate Searching Strategy

Assume for now that a model is available for the system described in the problem statement. Consider how a systematic search procedure might be designed for intelligent use of the model. For a given number of ovens purchased, the influence of "assemblers hired" will follow the pattern in Figure 1. With too few assemblers, profits are low because fewer widgets are made than could be sold. With too many assemblers, profits are low because not all widgets which could be made can be sold, and yet salaries must nonetheless be paid.

Similarly, for a given number of assemblers hired, average daily profit can be expected to vary with "ovens purchased" as shown in Figure 2. Profits are low with too few ovens, because not all widgets that could be sold can be made. At the other extreme, profits are low when there are more ovens available than needed to manufacture widgets at the rate at which they can be sold.

A three-dimensional perspective of the situation is suggested in Figure 3. Note that the mesh points in the grid laid down in the Figure 3 "hired/purchased" plane represent feasible (integer-valued) choices of values for the decision-variable pair. Located above each of these mesh points, in the third dimension, is the expected value of daily profit corresponding to that particular hire/purchase alternative (these points are not shown in Figure 3). The collection of all such cost points forms a "cost surface" in the third dimension [c]. Figures 1 and 2 are then simply contours through this cost surface.

It is easy to argue that the profit surface is unimodal, i.e., has a single maximum. The search for the optimal hire/purchase combination, then, is essentially a search for the only "highest point" on the profit surface. The problem is to establish an intelligent strategy for finding this point.

Consider this strategy.
(a) Estimate the average daily profit for some first choice of hire/purchase combination. (Preliminary analysis usually suggests what a good first choice would be.)
(b) Holding the "ovens purchased" constant, vary the "assemblers hired" until the high point in the corresponding contour has been found. This amounts to finding the high point in Figure 1. If this is the first time (b) has been performed, or if in performing (b) a higher profit than the previous best is found, go to (c); otherwise, go to (e).
(c) Holding "assemblers hired" constant at its high-point value from (b), vary the "ovens purchased" until the high point in that contour has been found. This is equivalent to finding the high point in Figure 2.
(d) Is the final hire/purchase combination in (c) the same as that in (b)? If not, return to (b); otherwise, go to (e).
(e) Call the best hire/purchase pair found so far the tentative optimum. As a final check, simulate for the as-yet-uninvestigated hire/purchase alternatives (if any) surrounding this tentative optimum. If one of them produces a higher profit, let this point be the new tentative optimum and return to (b). Otherwise, the search is finished.

For a numeric example which illustrates the approach, assume that the profit surface takes the form shown in Table 3. Assume further that the analyst starts with "23,7" as his first choice of hire/purchase combination. A history of his investigation is given in Table 4. (For a similar example in a different problem context, and a corresponding FORTRAN model designed for interactive use, see reference [3].)

[a] Assume the times are uniformly distributed over the indicated interval of integers.
[b] The durations of simulation required to move through transient conditions, and to gather meaningful steady-state statistics, usually must be determined by experimenting with a model of the system. The "1.5" and "10" day figures given in the problem statement were arrived at in such fashion.
[c] The points, rather than forming a "surface", are simply a collection of discrete values with no continuum joining them. It is nevertheless convenient to use the term "surface".

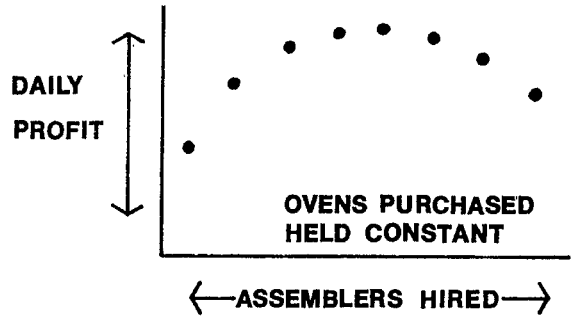

Figure 1. Dependence of Daily Profit on Number of Assemblers Hired

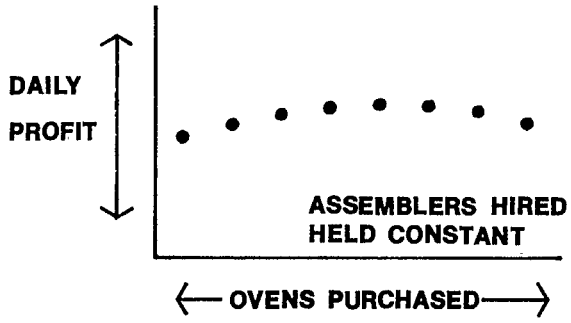

Figure 2. Dependence of Daily Profit on Number of Ovens Purchased

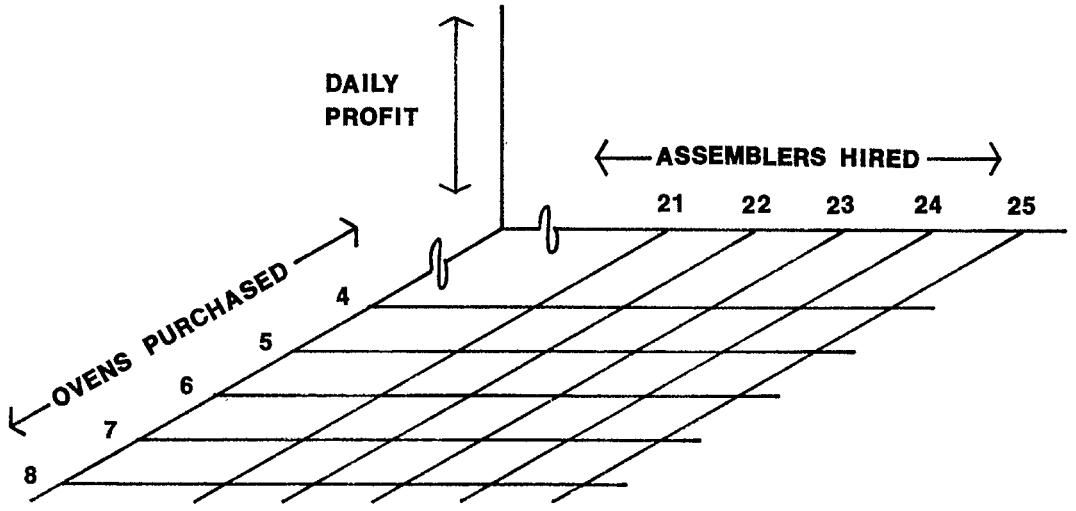

Figure 3. A Three Dimensional Perspective of the Daily Profit Surface

Ovens Purchased	Assemblers Hired: 21	22	23	24	25
4	$242	216	193	164	136
5	266	295	285	255	225
6	204	235	205	170	130
7	133	175	125	85	65
8	53	95	88	72	59

Table 3. Variation of Average Daily Profit, Dollars, with System Configuration

Simulation Number	Step in Strategy	Hire/Purchase Combination	Profit Estimate	Comments
1	a	23,7	$125	
2	b	24,7	85	
3	b	22,7	175	In the "7 purchased" contour, "22 hired" is found to produce the highest profit.
4	b	21,7	133	
5	c	22,8	95	
6	c	22,6	235	
7	c	22,5	295	In the "22 hired" contour, "5 purchased" is found to produce the highest profit.
8	c	22,4	216	
9	b	23,5	285	In the "5 purchased" contour, "22 hired"
10	b	21,5	266	is still the best combination.
11	e	23,6	205	The two alternative combinations neighbor-
12	e	21,4	242	ing "22,5" are now investigated.

The tentative optimum, "22,5", is more favorable than any of the neighboring alternatives. It is therefore optimum, and the search is finished.

Table 4. History of the Univariate Search Over the Table 2 Surface, Starting at "23,7"

The strategy just illustrated, then, enables the analyst to move systematically across the profit surface until he arrives at the maximum value of the objective function. From each "best point so far", moves are made along a contour only in the direction of increasing profit. When the high point in a contour is found, the next moves are made along an orthogonal contour. Eventually, a point is reached from which any move corresponds to a smaller profit. Assuming unimodality, this point is then the optimal profit.

In the example, each decision variable was either incremented or decremented by 1 when a move was being made along one of its contours. If the starting point is far from optimal, many simulations may then be required before the search reaches the neighborhood of optimality. A better approach is to start the search with relatively coarse increments for the decision variables. It is then possible to arrive at the region of optimality with relatively few simulations. When the "optimal point" corresponding to the first set of increments has been found, the first cycle is complete. From this "optimal point", a second set of (smaller) variable increments can be used to continue the search in a higher-resolution neighborhood. When this next "optimal point" has been located, the second cycle is complete. The increments can now be decreased a third time, and a third cycle performed, etc. When the last cycle is performed, the increment used for each variable will presumably be unity. Then, the "optimum point" found on the last cycle will be the true optimum, under the unimodality assumption.

A potential problem with the univariate searching strategy in the context of Monte Carlo simulation is that each simulation only provides an estimate of a profit surface value. If the variability of the estimator is high (because, for example, sample size is too small), a false conclusion can be drawn about the optimal system configuration.

Furthermore, if the surface is not unimodal, the univariate strategy will very likely only find a local maximum. In this event, the search must be conducted a number of times, using a different starting point each time. The various local maxima can then be compared by the analyst to identify the largest one. There is no guarantee, of course, that the largest entry in a possibly incomplete set of local maxima is the global maximum.

4. The FORTRAN Subroutine SERCH

The searching strategy described in Section 3 consists of a well-defined, step-by-step procedure capable of being programmed. Schmidt and Taylor have presented a FORTRAN main program which implements most features of this strategy [2]. The Schmidt and Taylor algorithm terminates, however, when the "tentative optimum" described at step (e) in Section 3 has been found. That is, it does not check diagonal neighbors of the tentative optimum. Hence, even if there is a single high point on the surface, the algorithm may not find it.

For the present application, Schmidt and Taylor's algorithm has been converted to a subroutine, and has been modified in several other major ways to decrease the execution time required to implement the univariate searching strategy. Because of its general utility, the source statements for this subroutine, named SERCH, are shown in Figure 4. A variable dictionary is given in Table 5.

SERCH can handle up to 20 decision variables. It requires these inputs.
(1) The number of decision variables (NDV).
(2) The number of cycles to be performed.
(3) For each decision variable, (a) the starting value, (b) the minimum feasible value, (c) the maximum feasible value, and (d) as a function of the cycle being performed, the amount by which to increment (decrement) that variable when the search is conducted along its contour.
(4) The estimate of the objective function's value corresponding to each decision variable vector for which a simulation has been performed. A "decision variable vector" is any particular set of feasible values the decision variables might assume. Note that the first decision variable vector, i.e., the "starting point", is inputted as data in (3a) above.

Inputs (1) through (3) occur the first time SERCH is called. They are not communicated to SERCH by the GPSS model, but are inputted by SERCH itself by reading from a data file. When these inputs are complete, SERCH places the first decision variable vector in fullword Savevalues 1 through 20 (or 1 through the number of decision variables, NDV) of the GPSS model, then returns control to GPSS. After the GPSS model has simulated for the system configuration corresponding to that vector, it calls SERCH, passing the objective function estimate in the call. SERCH then conducts tests to determine what next decision variable vector holds promise for improving the objective function's value. This next vector is placed in Savevalues 1 through NDV, and SERCH then returns control to GPSS, etc. Eventually, when the largest value has been found for the objective function at the last cycle, SERCH indicates this at the printer, then returns control directly to the operating system.

As suggested by the inputs at (3b) and (3c) above, SERCH limits the search domain to values which the analyst indicates are feasible.

Before SERCH requests a simulation of the GPSS model, it checks to determine that no simulation has yet been performed for the decision variable vector in question. If an estimate of the objective function's value at that point is already available, the simulation is not repeated. This has the advantages of (a) decreasing execution time by avoiding redundant simulation, and (b) avoiding the possibility of a no-convergence condition because of variability in the estimator.

Further description of SERCH will not be included here. By making use of the variable dictionary and comments cards in the program listing, those interested can come to a detailed understanding of the algorithm by studying Figure 4.

```
      SUBROUTINE SERCH(RESULT)
C
C  DECLARE MODES, DIMENSIONS, AND INITIAL VALUES.
      IMPLICIT INTEGER(A-Z)
      DIMENSION DV(20),DVMAX(20),DVMIN(20),DVOLD(100,20),DVSAVE(20),
     1 DVSTEP(20,20)
      DATA CALLNO,COUNT,CYCNO,DVMULT,DVNOW,MAX/-1,0,1,1,1,-999999/
C
C  TEST FOR FIRST CALL ON THE SUBROUTINE
      IF(CALLNO .NE. -1) GO TO 80
C
C  FIRST CALL:  READ IN AND PRINT OUT DECISION VARIABLE SPECIFICATIONS
      CALLNO = 0
      READ(5,10) NDV,CYCTOT
 10   FORMAT(2I2)
      DO 30 I=1,NDV
      READ (5,20) DV(I),DVMIN(I),DVMAX(I),(DVSTEP(K,J),J=1,CYCTOT)
 20   FORMAT(3I6/20I4)
 30   DVSAVE(I) = DV(I)
      WRITE(6,40) (I,I=1,NDV)
 40   FORMAT('1NUMBER OF   BOUNDS   INITIAL  STEP SIZES FOR CYCLES . . .
     1 '/'  VARIABLE  MIN  MAX   VALUE '20I5)
      WRITE(6,45)
 45   FORMAT(' ')
      DO 50 I=1,NDV
 50   WRITE(6,60) I,DVMIN(I),DVMAX(I),DV(I),(DVSTEP(I,J),J=1,CYCTOT)
 60   FORMAT(I6,I9,I5,2I7,19I5)
      WRITE(6,70) (I,I=1,NDV)
 70   FORMAT(///' RESULTS:'/'0OBJECTIVE  VALUES FOR VARIABLES . . .'/
     1 '  FUNCTION'20I5)
      WRITE(6,45)
      GO TO 160
C
C  NOT FIRST CALL:  STORE DECISION VARIABLE VALUES AND PRINT THEM WITH RESULTS
 80   CALLNO = CALLNO + 1
      DO 90 J=1,NDV
 90   DVOLD(CALLNO,J) = DV(J)
      WRITE(6,100) RESULT,(DV(I),I=1,NDV)
100   FORMAT(I8,I7,19I5)
C
C  BEGIN THE SEARCH FOR THE NEXT POINT TO TEST
      IF(RESULT .LT. MAX) GO TO 110
C
C  CURRENT POINT IS AS GOOD OR BETTER THAN THE PREVIOUS BEST.  SAVE THE
C  RESULT.  UPDATE COMPONENT IN SAVED VECTOR. CONTINUE SEARCH IN SAME DIRECTION.
      COUNT = 0
      MAX = RESULT
      DVSAVE(DVNOW) = DV(DVNOW)
      DV(DVNOW) = DV(DVNOW) + DVMULT*DVSTEP(DVNOW,CYCNO)
      GO TO 140
C
C  NEXT SECTION APPLIES IF CURRENT POINT IS NOT AS GOOD AS THE PREVIOUS BEST,
C  OR IF THE TENTATIVE NEXT POINT VIOLATES A BOUNDARY CONDITION, OR IF A
C  SIMULATION HAS ALREADY BEEN PERFORMED FOR THE TENTATIVE NEXT POINT.
C
C  STATEMENT 110 RESTORES THE VARIABLE WHOSE NEXT VALUE WAS NOT OF INTEREST.
C  (THOUGH NOT ALWAYS NECESSARY, IT IS NEVER WRONG TO DO THIS.)
110   DV(DVNOW) = DVSAVE(DVNOW)
      COUNT = COUNT + 1
      IF(COUNT .EQ. 2*NDV) GO TO 130
      IF(DVMULT .EQ. -1) GO TO 120
      DVMULT = -1
      GO TO 125
120   DVMULT = 1
      DVNOW = MOD(DVNOW,NDV) + 1
125   DV(DVNOW) = DVSAVE(DVNOW) + DVMULT*DVSTEP(DVNOW,CYCNO)
      GO TO 140
C
C  ALL ORTHOGONAL NEIGHBORS ARE INFERIOR TO THE CURRENT POINT.  CURRENT CYCLE
C  IS FINISHED.  BEGIN NEXT CYCLE NOW, UNLESS ALL CYCLES HAVE BEEN PERFORMED.
130   IF(CYCNO .EQ. CYCTOT) GO TO 170
      COUNT = 0
      CYCNO = CYCNO + 1
      DVMULT = 1
      DVNOW = 1
      DV(1) = DVSAVE(1) + DVSTEP(1,CYCNO)
```

```
C  TEST FOR POSSIBLE BOUNDARY CONDITION VIOLATION.
140   IF(DV(DVNOW) .LT. DVMIN(DVNOW) .OR. DV(DVNOW) .GT. DVMAX(DVNOW))
     1 GO TO 110
C
C  FIND IF A SIMULATION HAS ALREADY BEEN PERFORMED FOR THE TENTATIVE NEXT POINT.
      DO 150 I=1,CALLNO
      DO 145 J=1,NDV
      IF(DV(J) .NE. DVOLD(I,J)) GO TO 150
145   CONTINUE
      GO TO 110
150   CONTINUE
C
C  PLACE VECTOR IN SAVEVALUES 1 THROUGH NDV, THEN RETURN TO GPSS.
160   CALL PTFSVL(1,DV,NDV)
      RETURN
C
C  ALL CYCLES HAVE BEEN PERFORMED.  THE OPTIMIZATION IS FINISHED.
170   WRITE(6,180)
180   FORMAT('0MAXIMUM HAS BEEN REACHED')
      WRITE(6,100) MAX,(DVSAVE(I),I=1,NDV)
      CALL SYSTEM
      END
```

Figure 4. Source Statements for the Subroutine SERCH

(Continued from the Preceding Page)

Variable	Definition
CALLNO	Counter for number of calls made on the subroutine
COUNT	Counter for number of neighboring points producing a result inferior to that of the best point so far
CYCNO	Number of the cycle currently being performed
CYCTOT	Number of cycles to be performed in the optimization
DV	Vector whose I-th component is the current value of the I-th decision variable, I = 1,2,3,...,NDV
DVMAX	Vector whose I-th component is the maximum permitted value of the I-th decision variable, I = 1,2,3,...,NDV
DVMIN	Vector whose I-th component is the minimum permitted value of the I-th decision variable, I = 1,2,3,...,NDV
DVMULT	Value indicates direction of movement along current contour; 1 means "increment the current variable"; -1 means "decrement the current variable"
DVNOW	Number of the decision variable along whose contour the search is now being conducted
DVOLD	Matrix whose I-th row contains a record of the decision variable vector for which the I-th simulation was performed, I = 1,2,3,...,CALLNO
DVSAVE	Decision variable vector which produced the best result so far
DVSTEP	Matrix whose J-th column contains the vector of decision variable increments to be used during the J-th optimization cycle, J = 1,2,3,...,CYCTOT
MAX	Largest value of the objective function so far
MOD	FORTRAN function which performs modulus division; MOD(ARG1,ARG2) is the value of ARG1 modulus ARG2
NDV	Number of decision variables
PTFSVL	A subroutine; CALL PTFSVL(I,M,N) causes N integer values to be copied from the vector M (in SERCH) to fullword Savevalues I,I+1,I+2,...,I+N-1 in the GPSS model
RESULT	Dummy variable through which the GPSS model communicates the objective function estimate to SERCH

Table 5. Variable Dictionary for the FORTRAN Subroutine SERCH

5. The GPSS Model

A variety of alternative GPSS models can usually be found which are valid analogs for a stated system. In the present case, the need to communicate with the subroutine SERCH must be taken into account while building the model. Two-way communication between a GPSS model and a FORTRAN subroutine can take place through halfword and fullword Savevalues, through the Parameters of the Transaction currently at the HELP Block in the GPSS model, and through HELP Block Operands B, C, D, E, and F [d,e]. For the application here, only the fullword Savevalues and the HELP Block B Operand are used for inter-program communication. The GPSS model passes the estimate of the objective function's value to SERCH through the HELP Block B Operand; and, as already indicated in Section 4, SERCH communicates the next system configuration to the GPSS program by placing the decision variable vector in fullword Savevalues 1 through NDV (1 through 2, in this two-variable problem).

The GPSS model is shown as a Block Diagram in Figure 5. A corresponding Table of Definitions showing interpretations for the various GPSS entities appears in Table 6. The model has been divided into two segments, one for the simulation itself, and another to control the GPSS-FORTRAN interface. These two segments have been labeled in Figure 5, and explanatory remarks have been placed adjacent to the various Blocks, making it easier to follow the model's logic.

Consider first the Simulation Segment. Each Transaction circulating in this segment represents an assembler. The Storage named OVEN simulates the ovens. The number of Transactions in the Simulation Segment, and the capacity of the Storage OVEN, are controlled from the Control Segment by a method to be described below. The Control Segment also controls the duration of a simulation. When a simulation is to begin, the proper number of Transactions is routed to the first Simulation Segment Block from the Control Segment. Each assembler these Transactions represent then assembles a widget ("ADVANCE 30,5"), queues for an oven ("ENTER OVEN"), uses the oven ("ADVANCE 8,2"), frees it ("LEAVE OVEN"), adds to the count of finished widgets ("SAVEVALUE 4+,1"), and transfers back to begin the next assembly if the simulation is not yet over ("GATE S 1,ASSEM"). When the simulation is over, the "gate" at the GATE Block is opened from the Control Segment, permitting assemblers to leave the model, checking out as they do ("SAVEVALUE 5+,1").

The Control Segment is run by a single Transaction. When the simulation begins, this Transaction enters the model ("GENERATE ,,,1"), then calls the subroutine SERCH ("HELP SERCH, X3"). While the control Transaction is "parked" at the HELP Block, the subroutine reads the previously-described data file (see items (1) through (3), Section 3), places the number of assemblers and number of ovens for the first configuration in Savevalues 1 and 2 (i.e., X1 and X2), respectively, and returns control to the GPSS model. Moving on from the HELP Block, the control Transaction then closes the "simulation done" gate in the Simulation Segment ("LOGIC R 1"), and prepares to establish the proper capacity for the oven. Because Storage capacities cannot be directly modified dynamically as a GPSS simulation proceeds, this approach is taken.

(1) The Storage capacity is defined as 50 for the entire simulation. Prior analysis shows that this is far in excess of the optimal number of ovens.

(2) From the LOGIC Block, the control Transaction moves into the Block "ENTER OVEN,R$OVEN". This has the effect of exactly filling the Storage.

(3) The Transaction then executes the Block "LEAVE OVEN,X2". This makes the remaining capacity of the Storage equal to the number of ovens to be used in the current configuration.

After the proper oven capacity has been provided, the control Transaction enters the Block "SPLIT UPDTE,X1". This causes the right number of assembler-Transactions to be brought into the model. Their Priority Level is boosted above that of the control Transaction [f], and they are then routed to the first Block in the Simulation Segment ("ADVANCE 30,5"). The control Transaction then waits for 1.5 work days, or 720 minutes ("ADVANCE 720"), to elapse, while the transient-condition simulation takes place. The finished-widgets counter is then set back to zero ("SAVEVALUE 5,0"), and the control Transaction waits for another 10 working days at "ADVANCE 4800". The ten-day profit is then computed and placed in Savevalue 3, the gate in the Simulation Segment is opened, and the control Transaction waits until all assemblers have been removed from the model ("TEST E X5, X1"). It then initializes the departure counter at 0 for the next simulation ("SAVEVALUE 5, 0"), and cycles back to the HELP Block, where the ten-day profit is passed to SERCH through Savevalue 3 (X3). The control Transaction then remains parked at the HELP Block while SERCH sets the next system configuration, and the process repeats itself.

Note how the constraint of an assumed maximum sales rate has been reflected in the model. In determining the ten-day profit, either the number of widgets actually made (Savevalue 4) or 2750, whichever is smaller, is used as the total production on which the computation is based. This either-or effect is accomplished through the Function MADE. If a given configuration did produce widgets at an average rate exceeding 275 per day, then, the model behaves as though production is controlled so

[d] This statement applies to the HELP Block implemented for GPSS/360, Version I, in The University of Michigan's operating system. Those using the HELP Block in other environments should consult the available documentation to determine the corresponding possibilities.

[e] The HELP Block A Operand is the name of the FORTRAN subroutine being called.

[f] As a result of the priority boost, assembler-Transactions have a higher "increment the depart counter and go home" priority at the end of a given simulation than does the control Transaction in testing for an empty factory. If this priority distinction were not made, circumstances leading to a "no next event in the system" error message could arise.

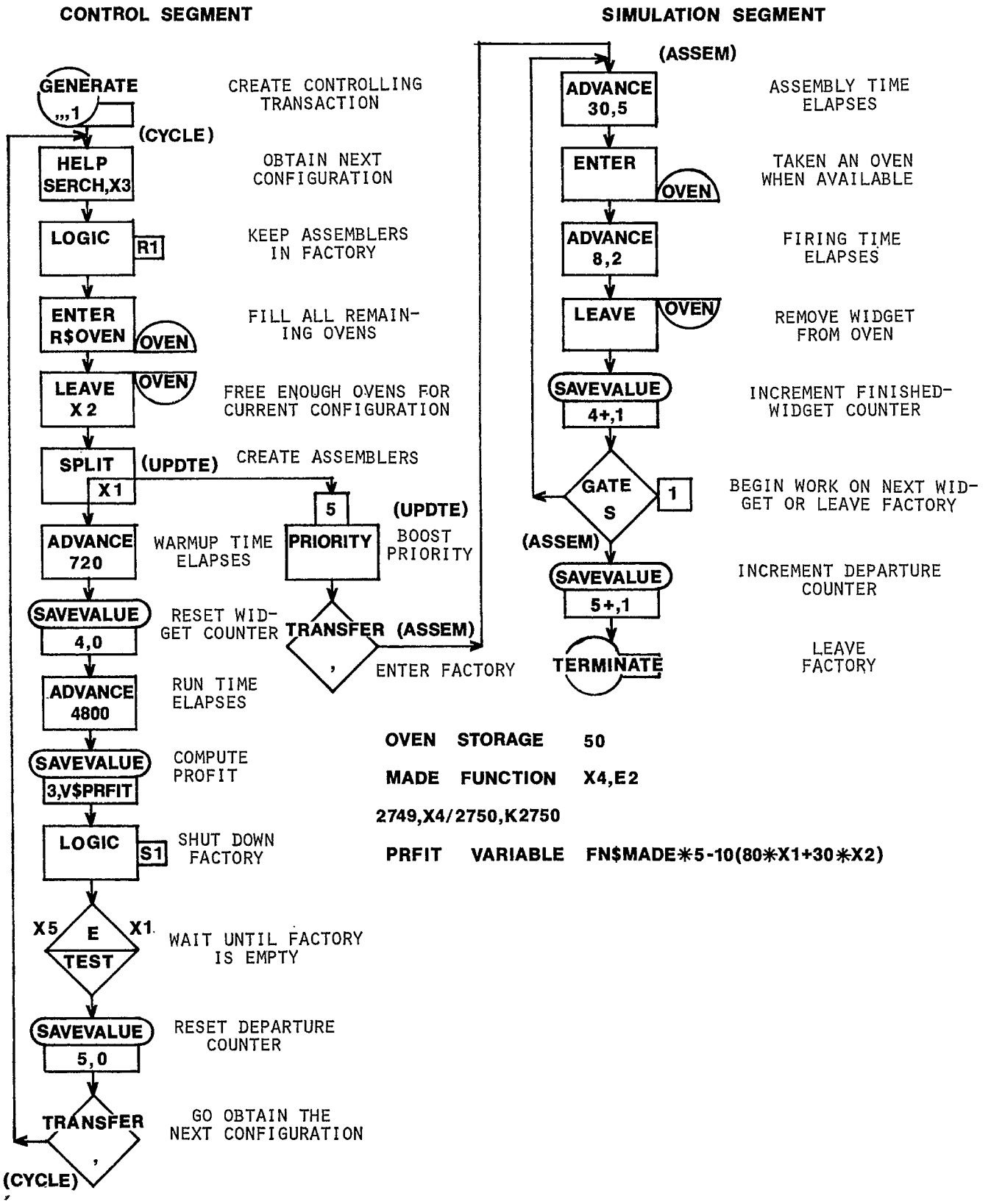

FIGURE 5. BLOCK DIAGRAM FOR THE GPSS MODEL

GPSS Entity	Interpretation
Time Unit: 1 Minute	
Transaction	
Control Segment	Supervisor who controls length of time the widget factory is open, as well as the number of assemblers and ovens in use
Simulation Segment	An Assembler
Function MADE	A Function whose value is either the number of widgets made during a 10-day simulation, or 2750, whichever is smaller
Logic Switch 1	A "gate" which keeps all assemblers in the factory until shutdown time
Savevalues	
1	Number of assemblers
2	Number of ovens
3	Total profit for the current configuration
4	Counter for finished widgets
5	Counter for assemblers leaving the factory
Storage OVEN	Storage whose available capacity equals the number of ovens in the factory (see Section 5 explanation)
Variable PRFIT	A Variable whose value is the profit associated with a 10-day simulation

Table 6. Definitions for the GPSS Model

as not to outstrip that rate. Of course, the profit computation still assumes that oven overhead and assembler salaries continue, even when production is deliberately held back to avoid making a product in quantities which cannot be sold.

Finally, note how fullword Savevalues 1 through 5 have been employed. From SERCH's point of view, fullword Savevalues 1 through 20 are used for the 20-component decision variable vector. In this application, however, there are only two decision variables. As a result, only Savevalues 1 and 2 are used in the decision variable context. SERCH knows this through the value of NDV, which is 2. Because Savevalues 3 through 20 are consequently available for other purposes, the GPSS model arbitrarily makes use of some of them. In particular, it uses Savevalue 3 to pass the profit estimate to SERCH; and it uses Savevalues 4 and 5 as a widget counter and departure counter, respectively.

6. Runs Made, and Results

A run was made for these conditions.
(1) The minimum and maximum feasible values for the number of assemblers were set at 0 and 50, respectively.
(2) The minimum and maximum feasible values for the number of ovens were set at 0 and 50, respectively.
(3) The initial decision variable vector was supplied as (23,7), i.e., 23 assemblers and 7 ovens. Preliminary analysis led to the choice of this pair of values.
(4) Two cycles were to be performed in seeking the optimal number of assemblers and ovens.
(5) Step sizes of 2 and 1 were specified for varying the number of assemblers during the first and second cycles, respectively.
(6) A step size of 1 was specified for varying the number of ovens during both the first and second cycles.

Output produced by SERCH for this run is shown in Figure 6. Simulations were performed for 16 different system configurations before the optimal combination of 22 assemblers and 5 ovens was found [g]. The various objective function values appearing in Figure 6 were used to construct Table 3 in Section 3. The path across the profit surface followed in the example in that section corresponds to the path followed when the model was run. The information in Figure 6 and Table 3 can consequently be compared. Note that Table 3 entries are profit per day, however, whereas objective function values in Figure 6 are ten-day profits. Also note that only one cycle was performed in the Section 3 example, using 1 as the increment for each variable.

Additional runs were also made under the conditions indicated above, except that the starting-point vector was varied. In several of these other runs, the simulation terminated before the optimal (22,5) point was reached. Analysis of the results indicated that this happened only because the "tentative optimum" (see step (e), Section 3) is accepted as the optimum by the Figure 3 algorithm. That algorithm should consequently be extended so that all immediate neighbors of the tentative optimum (and not just those in directions paral-

[g] Execution time for this run on The University of Michigan's 360/67 multiprocessing system was 142 CPU seconds. For comparison, another run was made with a GPSS model consisting only of a single GENERATE Block, then a HELP Block to make the first call on SERCH, then 16 additional HELP Blocks to return, respectively, the 16 profit estimates shown in Figure 6. CPU time for that run was 6.8 seconds. This comparison emphasizes the extent to which the simulations themselves use CPU time, in contrast with the time requirements of the univariate searching strategy.

```
NUMBER OF    BOUNDS    INITIAL    STEP SIZES FOR CYCLES . . .
 VARIABLE   MIN  MAX    VALUE       1    2

    1         0   50      23        2    1
    2         0   50       7        1    1


RESULTS:

OBJECTIVE  VALUES FOR VARIABLES . . .
 FUNCTION     1    2

    1250     23    7
     650     25    7
    1335     21    7
     680     19    7
     530     21    8
    2045     21    6
    2665     21    5
    2420     21    4
    2850     23    5
    2250     25    5
    2050     23    6
    1930     23    4
    2550     24    5
    2955     22    5
    2350     22    6
    2165     22    4

MAXIMUM HAS BEEN REACHED
    2955     22    5
```

Figure 6. Computer Output for a Test Run

lel to the coordinate planes) are examined before the search is stopped.

7. Summary

This paper, presented in the spirit of a tutorial, shows through a detailed example how a GPSS model can be interfaced with a FORTRAN subroutine, thereby overcoming some of the limitations inherent in the GPSS language. Some of the advantages in using FORTRAN with GPSS are those of (a) greater arithmetic capability, (b) availability of FORTRAN-based input and output, and (c) implementation of high-overhead activities, such as searching and testing, at execution times substantially lower than would be required if the same logic were performed in GPSS.

The example selected for this paper indicates in particular how an external optimizing algorithm, the univariate searching strategy, can be FORTRAN-implemented and then used with a GPSS model to search for the way to best configure a system. Enough details have been presented so that others can use the same FORTRAN subroutine in conjunction with GPSS models to move in the direction of automating the search for optimal system configurations. Even when the surface being searched is not unimodal in character, some or all of its local extrema can be identified by making a series of runs with a variety of starting points.

8. Biographies

Robert M. Lefkowits is associated with Arthur Andersen & Company as a member of the Administrative Services staff. He specializes in computer consulting, with emphasis on internal facilities management. He holds an A.B. in Mathematics from Dartmouth College (1969) and an MBA in Statistics and Management Science from The University of Michigan (1971).

Thomas J. Schriber is an Associate Professor of Statistics and Management Science at The University of Michigan. He regularly teaches an intensive one-week short course on GPSS in The University of Michigan's Engineering Summer Conference Series, and is a 1972 ACM National Lecturer on the subject. He is author of a book on GPSS (A GPSS Primer; John Wiley & Sons, Inc.; forthcoming mid-1972).

9. References

[1] Beveridge, G.S.G., and R.S. Schechter, Optimization: Theory and Practice, (McGraw-Hill, 1970).

[2] Schmidt, J.W., and R.E. Taylor, Simulation and Analysis of Industrial Systems, (Richard D. Irwin, Inc., 1970).

[3] Schriber, T.J., "Three Computer Models for Probability Applications", FORTRAN Applications in Business Administration, Vol. II, pp. 371-422 (The University of Michigan, 1971); copies of this article are available from T. Schriber on request.

[4] Wilde, D.J., and C.S. Beightler, Foundations of Optimization (Prentice-Hall, 1967).