8. MODELS WITH UNEQUAL INTERVAL TIMES. DISCRETE EVENT SIMULATION

8.1 Problems with continuous simulation. In certain problems continuous simulation, in which the **simulation clock advances with equal time intervals** and **all variables are supposed to change in each interval**, may result inefficient. For example, the model of a queue in a bank window can be programmed by such a method. At each interval time it must be asked if a new arrival or end of service will occur in the interval. The interval at which the event will occur must have been scheduled before. But these events happen at random times, may be separated by small or large time intervals. If the adopted fix interval is large, many events may happen in the interval and the result of the simulation may depend of the order in which these events are considered. If the interval is made very small to avoid this problem, then many intervals will be considered in which nothing happen. And this is not only a waste of time, but it is also unnecessary. In fact, when the future event is scheduled for a future interval, the exact time of the future event occurrence is known and there is no need to find it by exploring successive intervals of time in which nothing happens. The solution is obviously, **after processing an event, to go to process the next one, advancing the simulation clock to the time of occurrence of this next event**. This tactic was adopted in 6.2.2 c) example 2. The method can be generalized to more complex cases. This idea is the base of **discrete event simulation**.

8.2 Discrete event simulation. The method applies to a system in which **the state variables change only at certain instant in the time (events)** and are otherwise constant. In the example 6.2.2 c) it was seen that the state variables (length of the queue, state of the clerk, and type and time of the next event) only change when an arrival or an end of serve take place. In more complex systems may be many types of events.

Example. In an airport, there are the following types of event: an airplane takes the runway, an airplane leave the runway, an airplane take the gate for the passengers, or it depart from the gate. If the model includes the passengers other events are possible: a passenger arrives to the queue in the ticket office, a passenger finish to buy the ticket, (it may be different events of this types in different ticket window), the passenger enters the waiting room, etc. The movement of baggage may also be considered given many other event types.

At each event some state variables may change and new events may be scheduled. For instance when an airplane reach the gate 1, the variable “gate 1 state” passes from the value “free” to the value “occupied” and the event “the airplane abandons gate 1” may be schedule for a future time. Many processes occur **during** the same time in an airport, but **if the description is possible in terms of events, these events may be sequentially ordered in time, and they can be executed one by one in a sequential computer**. For example during the time in which the plane runs through the runway a passenger may be buying a ticket. These are simultaneous processes and cnnot be taken into account by a sequential computer. But the times in which the plane takes and leaves the runway and the times in which the passenger start and finish the buying of the ticket are separated and can be ordered in the time. So, they may be considered sequentially by the simulation algorithm in a sequential computer.

If two of them must occur **at** the same time, some priority criteria must be given to decide what happens first. An usual criteria is to execute first the one that was first scheduled for that time.

The system analysis consists in finding the events and determine, for each event, what variables must change and what new events must be scheduled. The general algorithm for the model is the following:

Read or define data:
initial values of the variables
in particular: Type and Time of initial events
system parameters
simulation parameters: TSim simulation time
Inicialization:
Accumulators for statistics to 0
TIME ← 0

Algorithm:
Next: IF TIME>TSim THEN GO TO **Final**
examine the next event (the one with lower time of occurrence).
See the Type and event time (Type =1,2,…N)
TIME ← event time
GO TO Type (1,3,..,N)

1: change the variables for event 1
schedule new future events
collect statistics
GO TO Next

2: change the variables for event 2
schedule new future events
collect statistics
GO TO Next
. . . . . .

N: change the variables for event N
schedule new future events
collect statistics
GO TO **Next**

Final: Write Statistics

STOP

The schedule can be made putting the "event element" : type of event , and time of future occurrence (EvTy, EvTi) in a **list ordered by increasing event time**. This is called FEL (**Future Event List**). To add an event it is necessary to intercalate the event in the FEL at the appropriated place to maintain the FEL ordered by increasing time (a new scheduled event may have a time less than a previously scheduled one). To take the next event the first of the FEL is taken.

Example. In an avenue of one way there are two traffic lights for pedestrians at a distance L. The part between the two lights is called the track.
The cars arrive at the traffic light 1 with exponential time arrival times with mean TBACar. If a car found the traffic light in Red or there is a queue (fixed or moving) or the track is full the car is added to the queue 1. Otherwise (light green, and not queue, and space in track) it pass the light and enters to the track. After this, it goes towards the next traffic light advancing the difference between L and the length of the queue at that traffic light. If there is a queue, or the light is red it is added to the queue 2. Otherwise it exit the system instantaneously.
The pedestrians arrive at the cross at times between arrivals with exponential distribution. If the light for the cars is red they cross the avenue and if it is green they do not cross. As it is assumed that the pedestrian know how much time the light will remain red they stop crossing before the lights turns green again, so a pedestrian that starts crossing with red light will arrive safely to the other side and will exit the system. If the pedestrians find their light red does not cross but the first one push a bottom to put the light for the cars red, which remain red for a certain time. The pedestrians do not form queue. After they put or find the traffic light red for the cars they exit the system immediately.
The physical length of the queue is their length in cars multiplied the length of a car. If the track is of only one lane this last is the physical length of the car plus a normal distance between cars. If there are n lanes this length must be divided by n, since each car adds (as an average) 1/n of that physical length.

ALGORITHM DESCRIPTION

Input data
State Variables:
Que1 ← 0 initial number of cars in queue 1 (queue 1 length in cars)
Que2 ← 0 initial number of cars in queue 2 (queue 2 length in cars)

TraLit1 ← Green initial status of traffic light 1
TraLit2 ← Green initial status of traffic light 2
Schedule car arrival at 1: Type ← 1 EvT ← 0 → FEL
Schedule pedestrian arrival at 1: Type ← 4 EvT ← 15 → FEL
Schedule pedestrian arrival at 2: Type ← 5 EvT ← 22 → FEL
Parameters of system:
TBACar ← 3.7 mean time between arrivals for cars
TBAPed1 ← 12.5 mean time between arrivals for pedestrians at 1
TBAPed2 ← 17.0 mean time between arrivals for pedestrians at 2
TimRed1 ← 15.0 time the signal for car remain Red for pedestrian crossing at 1

TimRed2 ←18.0 time the signal for car remain Red for pedestrian crossing at 2
L ←80 length of the track
CarLen ←2.8 length of a car (divided the number of lanes in the track)
Vel ←10 speed of the cars
Parameters of simulation:
TSim ← 7200 total simulation time

Initialization
SumQue1 ←0 to accumulate the sum of: length of queue 1×the last interval
in which the queue 1 remained unchanged
SumQue2 ←0 to accumulate the sum of: length of queue 2×the last interval
in which the queue 1 remained unchanged
Sum2Que1←0 to accumulate the sum of: square of length of queue 1×the last interval
in which the queue 1 remained unchanged
Sum2Que2←0 to accumulate the sum of : square of length of queue 2×the last interval
in which the queue 1 remained unchanged

LasCha1 ←0 to keep the time of the last change in the queue 1
LasCha2 ←0 to keep the time of the last change in the queue 2
MaxQue1 ←0 to keep the maximum of queue 1
MaxQue2 ←0 to keep the maximum of queue 2
TIME ←0

Processing the events:
NexEve: IF TIME≥TSim THEN GO TO **Final**

Extract the First Event (Ev,Ti) of FEL. Put time of the event time in: EvTim, event type in: EvTyp

TIME←EvTim update the simulation clock putting in it the time of the new event

NextEvent:

GO TO EvTyp: jump to event number (1,2,3,…9)

1. arrival car to traffic light 1
Schedule Next Arrival at traffic light 1: EvTy←1, EvT=TIME+EXPO(TBACar)
IF Que1=0 AND TraLit1=Green AND Que2×Carlen<L if path is free
THEN send car to track
Schedule Next arrival at traffic light 2 EvTy←2
EvTi←TIME+(L-Que2×CarLen)/Vel
ELSE if path is not free increase the queue
accumulate the values of queue length×the interval in which that length remained
P←Que1×(TIME-LasChaT1) SumQue1←SumQue1+P
Sum2Que1←Sum2Que1+P*P
LasChaT←TIME update time of last change

Que1 ← Que1+1 update queue 1
IF Que1>MaxQue1 THEN MaxQue1 ← Que1 update the maximum of Que1

2. arrival car to traffic light 2

IF Que2=0 AND TraLit2=Green if path is free
THEN
Schedule Next exit car: EvTy ← 3 EvT ← TIME
ELSE if path is not free increase the queue
accumulate the values of queue length×the interval in which that length remained
P ← Que2×(TIME-LasChaT2) SumQue2 ← SumQue2+P
Sum2Que2 ← Sum2Que2+P*P
LasChaT2 ← TIME update time of last change
Que2 ← Que2+1 update queue 2
IF Que2>MaxQue2 THEN MaxQue2 ← Que2 update the maximum of Que2

3. exit car

accumulate the values of queue length×the interval in which that length remained
P ← Que2×(TIME-LasChaT2) SumQue2 ← SumQue2+P
Sum2Que2 ← Sum2Que2+P×P
LasChaT2 ← TIME update time of last change
Que2 ← Que2-1 update queue 2

4. arrival of pedestrian at traffic light 1

Schedule Next pedestrian arrival at traffic light 1 EvTy ← 4 EvTi ← TIME+EXPO(12)
IF TrLi1=Green
THEN first pedestrian put the traffic light red
TrLi1 ← Red
Schedule Next light change: EvTy ← 8 EvTim ← TIME+TimRed1
Schedule next exit of pedestrian at 1 EvTy ← 5 EvTim ← TIME

5. exit of pedestrian from traffic light 1

6. arrival of pedestrian at traffic light 2

Schedule Next pedestrian arrival at traffic light 2 EvTy ← 6 EvTi ← TIME+EXPO(15)
IF TrLi2=Red
THEN
TrLi2:=Red
Schedule Next light change: EvTy ← 9 EvTim ← TIME+TimRed2
Schedule next exit of pedestrian at 2 EvTy ← 7 EvTim ← TIME

7. exit of pedestrian from traffic light 1

8. change of traffic light 1

TrLi1:=Green
IF Que1>0 AND Que2×CarLen<L if there are cars in queue 1 and path is free
THEN

Schedule Next arrival at traffic light 2 EvTy ← 2
EvT ← TIME+(L-Que2×CarLen)/Vel
accumulate the values of queue length times the interval in which that length remained
P ← Que1×(TIME-LasChaT1) SumQue1 ← SumQue1+P
Sum2Que1 ← Sum2Que1+P×P
LasChaT1 ← TIME update time of last change in queue 1
Que1 ← Que1-1 update queue 1

9. change of traffic light 2

TrLi2 ← Green
IF Que2>0 there are cars in queue 2
THEN
Schedule Next Event exit car EvTy ← 3 EvT ← TIME
accumulate the values of queue length times the interval in which that length remained
P ← Que2*(TIME-LasChaT2) SumQue2 ← SumQue2+P
Sum2Que2 ← Sum2Que2+P*P
LasChaT2 ← TIME update time of last change in queue 2
Que2 ← Que2-1 update queue 2

GO TO **NexEve**

Final: compute and write statistics

AveQue1 ← SumQue1/TSim;
DevQue1 ← sqrt((Sum2Que1-SumQue1*SumQue1/TSim)/TSim);

AveQue2 ← SumQue2/TSim;
DevQue2 ← sqrt((Sum2Que2-SumQue2*SumQue2/TSim)/TSim);

WRITE (AveQue1,DevQue1,MaxQue1, AveQue2,DevQue2,MaxQue2)

Procedures and functions

Procedure to insert an event element into the FEL ordered by time

The first element of FEL is pointed by PFirst. If the FEL is void it contains NIL
The values of the type and time for the event are given in the variables EvTy and EvTi
Create the element with fields EvTyp, EvTim, PointNext pointed by PNew:
PNew → (EvTyp, EvTim, PointNext)
PNew.EvTyp ← EvTy pass the given type to the element
PNew.EvTim ← EvTi pass the given time to the element
PNew.PointNext ← NIL the pointer to the next points now to no element

Put the new generated element in the FEL
IF PFirst =NIL

```
THEN  the FEL is void put the generated element as the first one
  PFirst ← PNew   now PFirst points to the new
ELSE      the FEL is not void, it must be searched to find the first with greater time
  PSearch ← PFirst   the pointer used to search points now to the first element of the FEL
  REPEAT WHILE   PSearch ≠ NIL   OR   EvTim>PSearch.Tim
     search while the searched is not the last of  FEL or it has a time less than the new
     PPrevious ← Psearch          keep the pointer to the actual before changing it
     PSearch ← PSearch.PointNext       now PSearch points to the following, repeat

  The repeat is finished because one with greater time is found or the FEL is exhausted
  if the search finished by finishing the list add the new after the last,  which is ponited
  by PPrevious
  IF PSearch=NIL
    THEN
      PPrevious.PointNext ← PNew  the last now points to the new
    ELSE  if the search finished because PSearch points to one with greater time than the new
          must be intercalate between the previous and the searched with higher value
          PNew.PointNext      ← PPrevious.PointNext    the new points to the searched
          PPrevious.PointNext ← PNew                   and the previous points to the new
End of the procedure
```

Procedure to extract the first of the FEL

```
 IF PFirst =NIL
   THEN WRITE( Error: Trying to Extract from FEL void)
   ELSE
     EvTy ← PFirst.EvTyp         pass event type to the variable EvTy
     EvTi  ← PFirst.EvTim         pass event time to the variable EvTy
     PFirst ← PFirst.PointNext    the first is now the following (or NIL)
  End of the procedure
```

Function that returns a random value from an exponential distribution with mean M

```
Function EXPO(M)
IF M ≤ ) THEN WRITE(Error: Exponential with non positive mean)
EXPO ← − M × ln(RANDOM)
Where random is a function that returns one (pseudo) random number from a distribution
 uniform in (0,1)
 End of the function
```

8.3 Problems with the event simulation. Process oriented simulation languages.

The program is rather complex in spite of many simplifications . The main difficulty is to get a correct and complete specification of the events. Besides, the program is not complete. The queues are nor really simulated, only a bookkeeping of their length is registered. To take into account the average delay of the cars in the queues, it would be necessary to represent the queues by lists of elements representing the cars, and register in each element the entry time and the exit time in each queue. The total delay in the system and the degree of crowding of the track may also be important. Other statistics, as the total number of cars and pedestrians generated may be

interesting in order to check the correctness of the program. The gathering of all this information made the program complex even in this trivial example.

Approaches to simplify the programming has been made since 1960. One important idea was to create a language in which the user can describe the successive changes that the movement of an element (like the cars in the example) through the system introduce in the system and in the characteristics of the element. With this information a compiler may create the program to do the simulation. This was successfully done by Gordon and others in the GPSS language. The language consists in a repertory of "blocks" of different types, and moving entities (in the GPSS they were called "transactions") that move from one block to other. The blocks can generate, move, stop, transform, and destroy the transactions. So the user describes only the process at which the moving entities are submitted and the changes in variables that these entities produce, which is more easy that the finding and specification of the events. The languages based in this idea are called **process oriented simulation languages**.

Example. Consider a bank teller that serves 1000 clients that come at random with interval times taking from an uniform distribution between 3 and 7 minutes. The service takes a random value between 1 and 7 minutes from a uniform distribution. The GPSS program (with comments of the meaning of each instruction) is as follow:

GENERATE	5,2,0,1000	generate 1000 transaction starting at TIME=0, with random inter-arrival times from 5-2 to 5+2
QUEUE	LINE	the transactions enter in a queue that was called LINE (it may be another name)
SIZE	TELLER	a facility, (that was called TELLER, and it is described by a variable that has two states: free, engaged), allows the transaction to pass only if it is free, then it is put in engaged
DEPART	LINE	when the transaction is allowed to pass it abandons the queue called LINE. The TELLER continues engaged
ADVANCE	4,3	a random value between 4-3 and 4+3 indicates the simulation time during which the transaction is held.
RELEASE	TELLER	after such a time the TELLER is put again free allowing other transaction in LINE (if any) to take the TELLER
TERMINATE		the transaction is destroyed
START	1	1 is subtracted from the number of transaction to be generated (originally 1000). When it is 0 the run stops.

This description (without the comments, of course) is read by a compiler that generates a simulation program (something similar to the above program) that makes the simulation. A complete statistics is recorded during the simulation and printed at the end including size of the queue (final, mean, deviation, maximum, minimum), the time during which the transactions remain in the queue (maximum, minimum, mean and deviation), the time of occupation of the facility (maximum, minimum, mean) the proportion of the simulation time during which the facility is engage, the number of transactions that enter each block, etc.

The language was endowed by a lot of facilities: typical distribution functions, possibility to attach values to the transactions (called parameters), many types of blocks that can stop transactions according to logical conditions, send the transactions to specified blocks, use and change the values of the parameters, synchronize the movement of transactions. Use of tables, facilities to generate reports, possibility to incorporate procedures written in FORTRAN gives to the language a great flexibility.
GPSS was used in many applications. Although it was superseded by more sophisticate languages, it was a model for many of the new languages like SLAM and SIMAN (see chapter 9). New versions of the GPSS for PC are described by Chisman 1992.

In Chapter 9 the GLIDER language will be introduced that allows to mix, in a direct and natural mode the continuous and discrete event simulation. In the GLIDER language the blocks are called **nodes**, and the transactions are called **messages**. The instructions for the above program in GLIDER language are:

```
Arrival (I)  ::  IT:=UNIF(3,7);       the type I node produces a message each IT time
                                       Units (IT  takes its value from a uniform distribution)
                                       and send it to the successor node (here the next one)
Teller   (R)  ::  STAY:=UNIF(1,7);  the type R node places the incoming messages  in a
                                       waiting queue, enters them one by one in an internal
                                       list and delivers them at times indicated by the STAY
Exit     (E) ::                        the type E node destroys the message sent to it by the
                                       Teller node
INIT TSIM:=200; ACT(Arrival,0);   simulation time and first activation
DECL STATISTICS ALLNODES;  statistics for all the nodes are required
```

EXERCISES

1. Translate the code of the traffic light program in 8.2 to a general purpose language (Pascal, C, Fortran) and made some experiments with it.
2. Describe the algorithm of a program for the following system using the discrete event approach :
Assembled TV sets are tested, which takes between 9 and 15 minutes, 12 minutes being the most likely delay. The unit entry the test section at random time intervals with a GAMMA distribution with mean 6 and deviation 1. In the test section 8% failed to pass the test and go to a reparation section where they remain a time with a GAMA distribution with mean 32 and deviation 9. In the repair section 12% of the received are definitely rejected and the other come again to the test section. Those rejected by a second time are definitely rejected. The test section has two technicians that work simultaneously. The repair section has only one.
Identify the events and indicate in natural language the changes of variables and schedule of new events in each event. Note than in this case it is necessary to simulate the moving entities because they transport a characteristic (if they have been repaired or not) that is changed in a part of the system and are used in other.

BIBLIOGRAPHY

Chisman J 1992. Introduction to Simulation Modeling Using GPSS/PC. Prentice Hall. Englewood Cliffs, New Jersey. (A practical introduction to simulation in a new version of this old language, enhanced by graphic facilities).

Schriber T.J. 1974. Simulation Using GPSS. Wiley, N.Y. (A complete and detailed exposition of the practice of simulation using GPSS, interesting yet for the quantity and quality of the examples).

GLIDER Development Group, 1991: GLIDER Reference Manual.
CESIMO & IEA Universidad de los Andes Venezuela.
Edited by Interdisciplinary Research Center. San Diego State University. College of Sciences.