GRAPHS FOR QUEUEING MODELS

B. Helbrough & G.T. Herman
IBM (UK) Ltd.
101 Wigmore Street
London, W.1.

Abstract

We discuss a program whose task is to produce graphical answers to problems arising in queueing theory. We consider why such a program is necessary and describe the situation to which it is applicable. Some of the statistical and programming considerations for the design of the program are mentioned. An example of its application is given.

1. MOTIVATION

Queueing theory is a comparatively recent but rapidly developing branch of applied mathematics. Its subject matter is the investigation of various queueing characteristics, such as the average queue length or waiting time in a traffic system, e.g. a cafeteria or a real time computer system. A collection of techniques and results in queueing theory and a shorter work on the more specialist real time system design aspects will be found in the references 1, 2.

In common with most other branches of applied mathematics, queueing theory deals with idealised situations. When applied to a real life problem it will generally only be able to provide approximate solutions. Because it is a young discipline, solutions are not available even for cases of great practical importance, e.g. for queueing models in which arrivals at the queues happen in a more or less clustered fashion. Such clustered arrival is typical in certain real time computer situations.

Even in situations where analytic solutions exist, their use is often time consuming. A general question which may arise in queueing situations is the following: how does the variation of a certain characteristic (e.g. servers utilisation) affect the variation of a certain other characteristic(e.g. average queueing time)? To answer such a question we need a graph which plots the queueing time against the server utilisation. For producing such a graph we may need to work out the average queueing time for server utilisations from 30% to 95% at 5% intervals, 14 calculations in all. In all but the simplest situation such calculations may take a very long time. It can be argued that such graphs should be preproduced (and they are indeed available for the most common cases), but the variety of situations which may arise make the production of a book containing all such graphs of possible importance impractical if not impossible.

The Graphs for Queueing Models program (GQM for short) has been developed, so that such graphs can be produced quickly for each situation as it arises. Because it is based on simulation rather than analytic techniques it can deal with a large variety of problems for which analytic solutions are not yet available.

2. THE BASIC MODEL

The GQM program is applicable to situations of the following type (see Fig. 1).

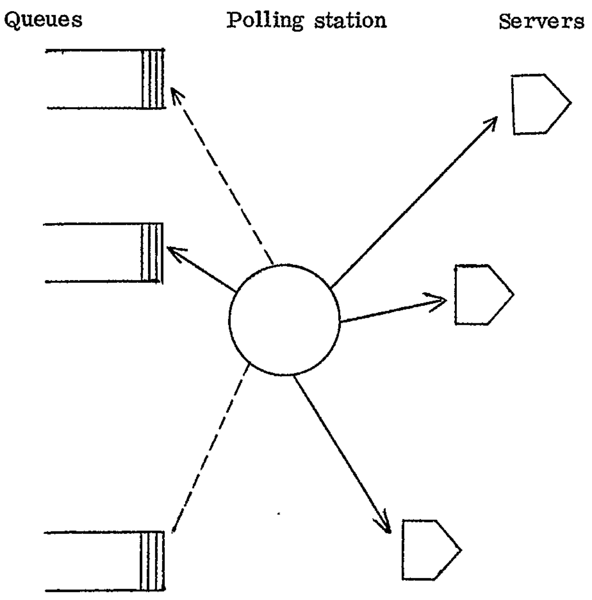


Fig. 1. The basic queueing model.

Transactions arrive at a number of queues each having its own arrival rate. They wait in the queue until the queue gets polled, then one of the transactions which has waited longest in the queue will go through the polling station to one of the available servers. (Firstin, first out queueing discipline). At the server it will be serviced for a certain length of time (depending on the transaction only and not on the server) after which it will leave the server and the system.

Polling will take place if, and only if, at least one of the servers is available. The time taken up by polling does not depend on which server is available. In fact, since we are basically interested in the behaviour of the queues rather than that of the servers, we do not distinguish between the servers and treat them as a pool. If more than one server is available we do not care which of them will service the transaction. However, the average utilisation of this server pool may be of interest.

When somebody wishes to make use of the GQM program, he will have to provide some data to make the model specific to his application. The kind of information that is required from him, and this also indicates the flexibility and range of applicability of the program is the following:

i) Number of queues.
ii) Number of servers.
iii) Interarrival distribution between clusters (e.g. time between clusters constant, or an exponential distribution, etc.).
iv) Cluster size distribution (this could be constant, in fact 1 if there is no clustering, or it could be any other distribution).
v) Distribution within cluster (e.g. all arriving together, normally distributed around the cluster mean, etc.).
vi) Service time distribution (e.g. constant, exponential).
vii) Polling discipline (e.g. random, sequential etc).
viii) Polling time distribution.
ix) The characteristic which is to form the argument of the graphs (e.g. server utilisation, the spread of clusters) and the number of arguments for which values are to be determined.
x) The characteristics which are to form the values of the graphs (e.g. mean queueing time at a given queue, maximum queue length at a given queue).

These definitions are quite flexible, e.g. the distributions in (iii) - (vi) can be made to depend on the queue at which the transaction originally arrives. Standard sets of data cards are provided for the most frequent situations.

Also, because the graphs are produced as a result of simulation, and hence essentially experimentation, the user is given freedom to determine the length of his experiments, the number of his experiments and the degree of confidence he wants to have in his results. We now turn to the discussion of these points.

3. STATISTICAL CONSIDERATIONS

In the running of the simulation model, the interarrival time, cluster size, service time, etc. will generally depend on the random number generator of the model. Hence there will be fluctuation in all the characteristics of the system and to calculate something like the mean queue length (if there is one) we need an infinitely long run. In practice we should choose our runs long enough to make exceptional situations insignificant.

In particular, since initially there are no queues at all, it is better to exclude the beginning of the simulation run from the calculations. The GQM program automatically excludes the first third of any run from its statistics gathering.

Even if we take very long runs and exclude the first third of each from the statistics, we cannot say for certain how confident we are of our results. This is because standard tests of confidence (e.g. t-test, χ^2 - test) assume independently collected samples, whereas the queue length at any given time is highly dependent on the queue length at previous times. The calculation of the level of confidence for a complicated queueing model can be a very difficult exercise indeed.

The GQM program eliminates the need for such calculation in the following way. It carries out a number of experiments, each starting from scratch, but generating different random numbers, ignoring in each experiment the first third of the run. The results of these experiments are now statistically independent and t-test can be applied. This is done automatically be the program. For each graph we obtain two possible values U and L indicating the upper and lower limits of the true value within the user specified degree of confidence. For example, if the user specifies 99% confidence level and a run length of 1500 transactions and he requires to know the maximum queue length, after looking at his two graphs (for U and L respectively) he can make the following statement: I am 99% confident that the average of the maximum queue length in a run of 1000 transactions after an initialisation period of 500 transactions is not more than U or less than L. He can say this, because the program has worked out that if the average of the maximum queue length was greater than U or less than L the kind of simulation runs which have happened could only happen on the average once out of every 100 cases.

If the difference between U and L is too large for the user's liking, he can decrease it either by lowering his level of confidence, or by increasing the number of runs or the length of the individual runs.

4. PROGRAMMING CONSIDERATIONS

The simulation language used in the GQM program is GPSS/360. This was chosen because of the excellent capabilities of GPSS to describe all kinds of traffic and queueing situations and because of the graphic output available with GPSS/360.

The data which the user has to provide for the program (described in Section 2) is put in front of the program deck in the form of variable cards, storage definition cards and savevalue initialisation cards. At this point definition of certain functions may also be necessary to describe distributions. Cards for standard situations are available. These cards are followed by the GQM deck, which includes all all the necessary GPSS control cards. (Length of run is determined by a variable operand in the terminate block).

Although the program is relatively short, its logic is fairly complicated, and not more than a brief mention of one of its basic features can be given here.

All the exprriments in the simulation are carried out in parallel. Hence there are N copies of the model working in the simulation run simultaneously, where N is the product of the number of arguments required for the final graphs and the number of experiments requested by the user for the determination of values for any of the arguments. This feature helps us in satisfying the user's requirement for the number of experiments and also in the production of the final graphs which depend on the results of all the experiments. Running N copies simultaneously is achieved by the use of split blocks at appropriate points in the program.

The program makes good use of some features of GPSS/360 which were not available in earlier versions of GPSS. Features used include graphic output, multiple storage definition cards, Boolean variables, matrix savevalues, and user chain SNAs.

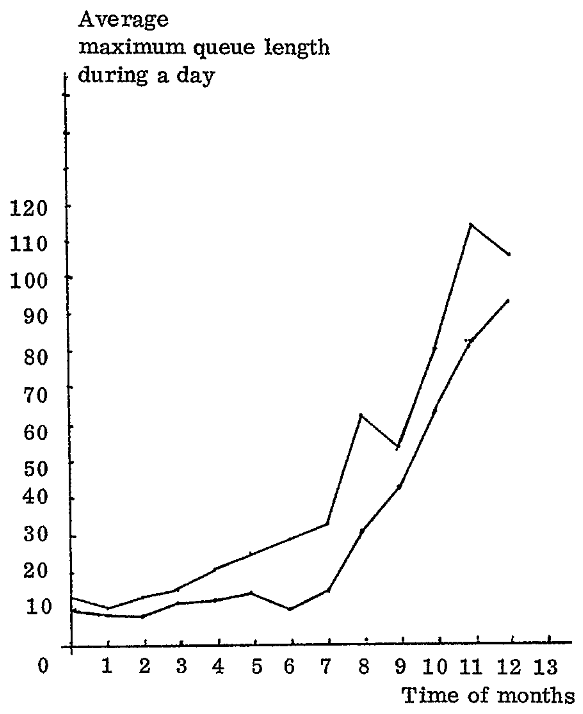
Because of our aim to provide an upper and lower limit for each value to be calculated (see Section 3) two graphs have to be produced for each characteristic under investigation. Since it appears advantageous to have both these graphs within the same axes (a task which is impossible for GPSS) and also in a way which is visually clearer than the GPSS/360 graphic output, we have additional feature of producing our graphs on the IBM 1130, using the Data Presentation System which is capable of plotting several graphs within the same axes. An example of this type of graphic output is discussed in the next section.

5. AN EXAMPLE

It is predicted that the traffic rate for a multidrop data communication line will increase by approximately 10% per month. We are anxious that the queue of messages waiting to be transmitted by the operator does not become unmanagably long. After investigation of the situation (method of message arrival, transmission time, polling discipline) we obtained by the help of the GQM program the following graph. This shows us, with 95% level of confidence, the limits within which the average of the maximum queue length during a day for the next year will lie.

References

1. SAATY, T.L., Elements of Queueing Theory, McGraw-Hill, New York, p.423, 1961.
2. Analysis of Some Queueing Model in Real - Time Systems, IBM Technical Publications Dept., (Manual F20-0007), p. 75.



VARIABLE MESH SIMULATOR

M. J. Kelly

IBM Electronics Systems Center
Federal Systems Division
Owego, New York 13827

The Variable Mesh Simulator is an experimental program running on S/360 under MVT OS. It is so named by analogy with the numerical solution of differential equations where finite difference approximations are systematically evaluated at discrete intervals of their variables. The largest intervals consistent with the accuracy of these approximations are used in order to minimize the amount of computation. In areas where greater precision is required, smaller intervals must be used and so we get a picture of an n-dimensional grid whose mesh-size varies according to local needs. In VMS, different language-levels of description of the system (to cater for different objectives) are our approximations, and these levels imply the degree of detail required -- our interval sizes. Also, by dividing the system into parts, independently described at different levels, we can concentrate our computer power on those areas needing the most intenstive study. Each such "part" is called a <u>region</u>, and each level of description of a region is called a <u>cut</u>. Our overall objective is, then, to be able to simulate economically any or all parts of a system, no matter how large, at any levels of description and for any purpose the user has.

A region of a system is represented as a connected set of "black-boxes" (called <u>blocks</u>) and the basic simulator is a program which takes them, after suitable preparation, and schedules <u>events</u> which reflect their behavior in simulated time. The descriptions of the blocks tend to be highly application oriented, and, therefore, special languages would be advantageous to the user. However, we have employed only Assembler macros and PL/1 to date although any other language can be used if a suitable compiler exists. As new blocks are defined, significant features are noted so that its compiled description can be made reasonably general without performance degradation. Gradually, a library of useful block descriptions is built up, which can be used in subsequent system studies.

Cuts are assembled so that the transfer of data during simulation does not require table searching. Each block is stored as a value, plus a list of addresses which are of four types -- data sources, data destinations, subroutine entries, and successor blocks. The <u>value</u> of a block denotes the current state of that part of the system; the values of other blocks, used as input data, constitute the block's environment since there are no other connections to the system. The state of a block changes at discrete points in simulated time and is considered steady between these points.

When control is given to a block, its subroutines process the input data and store the results as block values before exiting to a standard routine which schedules events for the successor blocks. Depending on what a block's function is, the value it generates, if any, may be its own or another block's. An example of the latter is a storage block which has value but no function, the transfer of data being due to a separate control block.

When a block's function has been simulated and any new values stored, it is common to require the operation of other blocks at the same or later points in simulated time. Each such event is entered in a queue for the required timing point. The number of such queues will vary during simulation as will the numbers of events in them. A simple technique is used to allocate space for these queue entries dynamically such that after execution, the space is made available for new entries.

The structure of these queues is essentially a two-dimensional linked chain. Events in a queue are linked forward; i.e., each entry contains the address of the next one. Each queue has a header which addresses the first and last events in the queue. The headers themselves are a forward and backward linked chain and contain the time value at which all events in their queue are to be executed. Although all events occurring at one time are, ostensibly, simultaneous, the computer can handle them only consecutively, so some attention has to be paid to the order in which they are scheduled. In practice, a binary priority scheme suffices. Two priorities, high and low, are recognized. Events scheduled high are entered in the queue on a LIFO (last in, first out) basis, and those low are FIFO. The system user has control now, since he can specify special ordering between closely related events.

The mechanism described belongs to one cut. Every cut having its own queues, the complete system is simulated by exercising the cuts individually until no further activity at the current time is scheduled, and this may involve resimulating cuts at that same time if instantaneous feedback occurs. It is at run time that the system model is put together. Not all the existing cuts need be used -- only those necessary for the immediate purposes. Up to this time, a cut has many dangling connections which are its undefined interface with the rest of the system, and only now need the connections be hooked up. This delayed specification permits great flexibility in system design since incomplete or alternative designs can be handled without trouble. The cut interfaces are specified by tables stating the line names and data formats of each. These interconnections are then assembled to form the internal linkage structure, which comprises special IN and OUT blocks, plus address lists for data transfers and successor scheduling. At a later date, when user intervention via a terminal is implemented, it will be possible to modify the intercut connections dynamically and, therefore, to switch in and out alternative descriptions of cuts. For example, if an engineer is simulating a machine and he locates an error in one part, he can continue simulating other details of his machine by using a higher level "functional" description of the faulty part. In a multiprogrammed computer, he can reassemble the corrected cut concurrently and then switch it back into his running model for further checking.

A problem with previous simulators, in spite of their relatively limited scope, has been capacity limitations due to core requirements. This is an inherent problem so we have planned a roll-out/roll-in capability to solve it, albeit at some cost in performance. To reduce quantities, block values and outstanding queues are the only data rolled out; and, to facilitate this, the cut assembler keeps block values separate from the address lists. On roll-in, both the rolled out data and a fresh copy of the remaining cut material is required. Since different cuts occupy different amounts of core, it is impossible to keep every cell in use, but the user can attempt to define his cuts with approximately equal assembled sizes. In addition, it is planned to permit fragmented cuts; i.e., subdivided cuts which have few interconnections between subdivisions and which are re-linked together at roll-in time if necessary. Here, as in aspects treated earlier, the on-line user and dynamic change capability are considered key concepts whose value will far outweigh their cost.

The program, while still under development, is being used to simulate computers described at logic block and register levels and modeled in 2- or 3-valued, unit or variable delay modes.

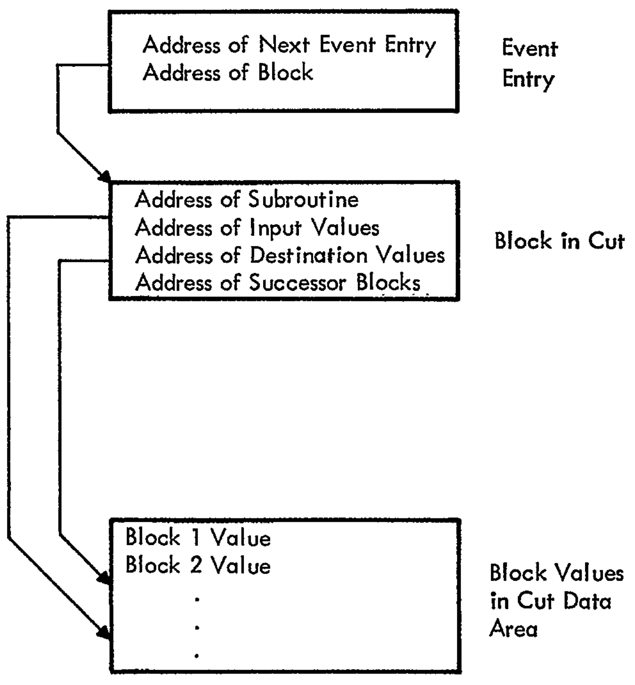

Address of Next Event Entry
Address of Block
Event Entry
Address of Subroutine
Address of Input Values
Address of Destination Values
Address of Successor Blocks
Block in Cut
Block 1 Value
Block 2 Value
Block Values in Cut Data Area

Figure 1. Standard Block Linkage

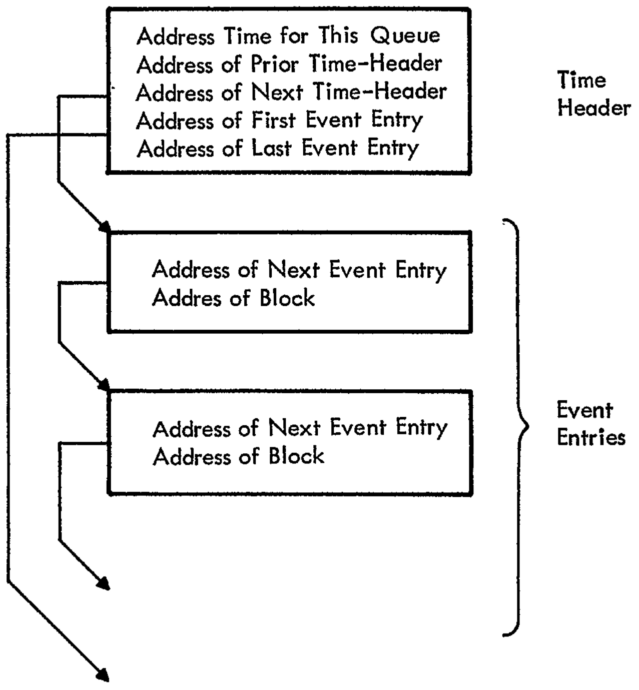

Address Time for This Queue
Address of Prior Time-Header
Address of Next Time-Header
Address of First Event Entry
Address of Last Event Entry
Time Header
Address of Next Event Entry
Addres of Block
Address of Next Event Entry
Address of Block
Event Entries

Figure 2. A Typical Time Queue

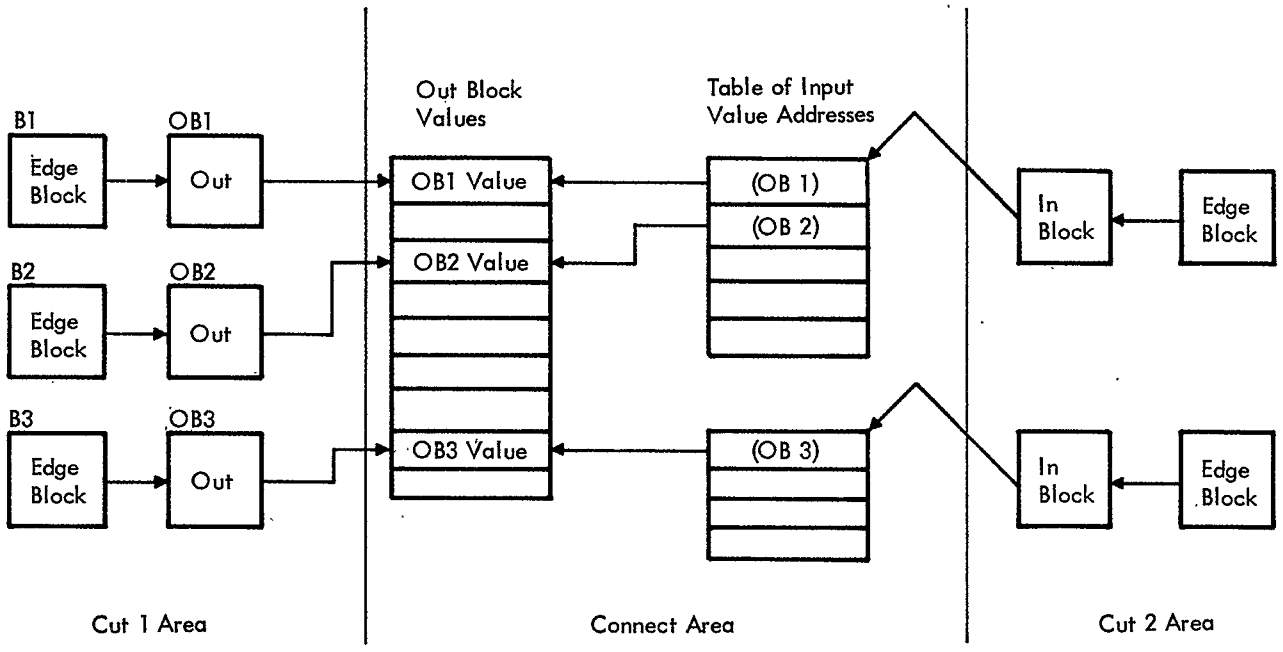


Figure 3. Cut 2 Accessing Block Values as Defined by the Connect Area

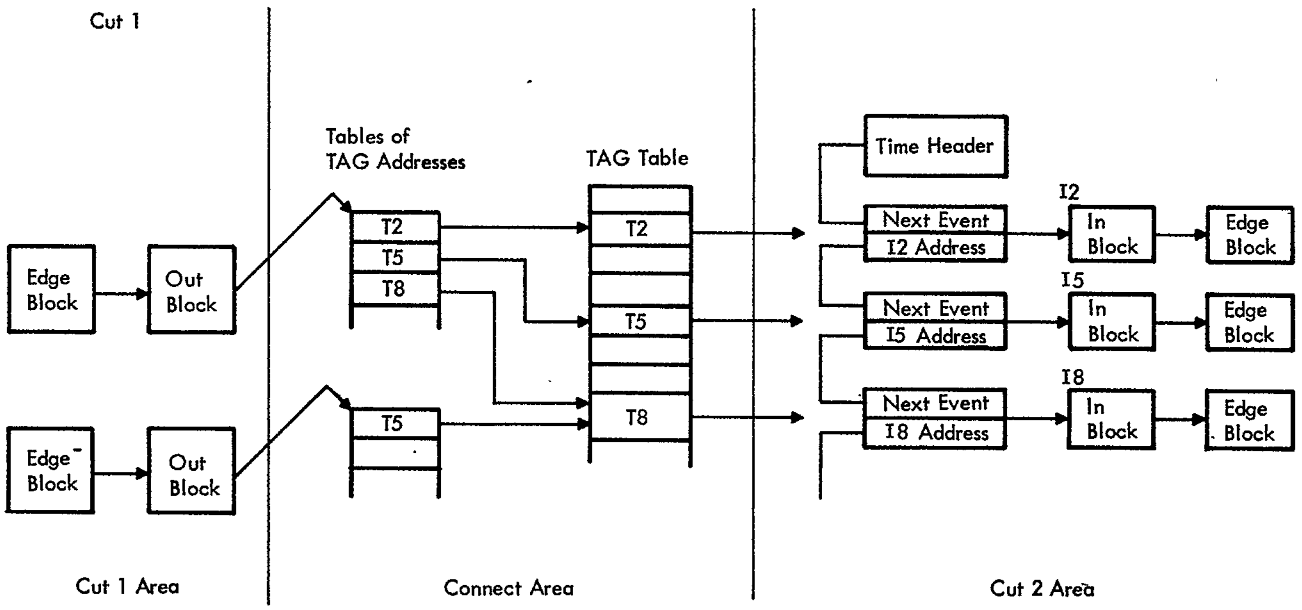


Figure 4. Cut 1 Stimulating Cut 2 By Passing Control Via the Connect Area