JOB SHOP SCHEDULING SIMULATIONS

FOR INTERACTIVE USE IN COMPUTER GRAPHICS

John W. O'Leary
Member of Research Staff
Western Electric Research Center
P. O. Box 900, Princeton, New Jersey 08540

Abstract

A discussion of an interactive computer graphics system used for job shop scheduling problems, with emphasis on the structure of the job shop simulator schedule generator. Three different language implementations are discussed in light of the special requirements of this application.

INTRODUCTION

Job Shop Scheduling remains a difficult area for analytical analysis. The combinatorial nature of the magnitude of alternative feasible schedules and conflicting measures of performance involved in the scheduling function make optimal solutions impossible for interesting job shops.

Due to the lack of closed mathematical solutions, simulation programs have provided convenient vehicles for development of heuristic scheduling. This paper discusses a job shop simulation program used to generate schedules which are displayed immediately on a computer graphics cathode ray tube in the form of a Gantt chart. Interactive communication then proceeds with the scheduler able to command the simulator to generate different schedules and receive manual changes. This allows schedulers to interject their own decision making into the scheduling process. Since it is felt that this use of interactive graphics can be of significant help in scheduling,* it is of interest to describe the structure of the special nature of the job shop simulator that is required for this project. Further, it is of interest to evaluate three important languages (FORTRAN, SIMSCRIPT II, GPSS/360) against specific criteria that arise due to certain unique tasks that are required of a programming language applied to this project.

This paper discusses the Graphic Job Shop Scheduling Project (GJSSP) and describes the characteristics of the schedule generator simulation program. The important differences in how each of the language implementations treat the job shop are mentioned in terms of certain criteria and their respective measures of performance. It should be mentioned that in each of the three language implementations NO attempt was made to optimize performance. That is, the techniques used in programming were such as to develop the model as quickly as possible, and once the program was debugged, no further sophistication was introduced. The assumption used is that typically most Operations Research Analysts are interested in quickly passing through the programming step of model building, and thus, they are willing to sacrifice computer time for speedier project completion. Further, the reader will have to assume an equivalent level of programming pre-experience in each language. These assumptions should not effect the discussion of the simulation shop model of GJSSP or its computer implementation.

GJSSP DESCRIPTION

The Graphic Job Shop Scheduling Project is implemented on a IBM 360/50 which communicates with a Digital Equipment PDP-9 computer that drives the graphics display. The project is shown schematically in chart 1.

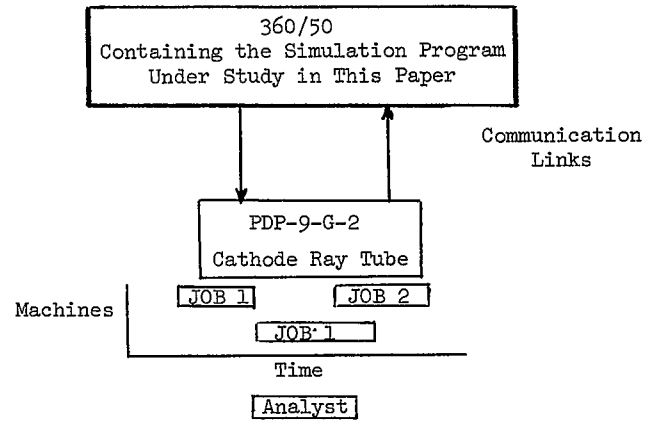


CHART 1

The following steps are typical of a successful run of the GJSSP. The goal represented in these steps is to schedule a given shop.

1. The Simulation Program is called from a Disk Library upon submission of a deck to the 360/50. Contained in the deck are cards that define the shop to be studied. These cards contain job and machine numbers, technological ordering of operations on machines, processing times and characteristics of the job such as due date, lateness cost, priority and shop arrival time.

* "Allegedly, human schedulers operating on some form of Gantt chart can employ various heuristic adjusting procedures... ." With the advent of graphical input-output devices for computers, it is now conceivable that a meaningful experiment of this nature may be performed.[1] Also see 3, 4.

2. The job is executed in low speed core. (IBM unit 2361, LCS 1000K storage). Although compute bound execution is about four times slower, the job is allowed to stay resident in low speed core during the whole time of interactive use. The program runs with high priority, but during zero demand of GJSSP, 360 execution of other programs is not effected.

3. Upon completion of execution, a vector is created that contains the information about the generated schedule (the initial schedule is always generated by the first come first serve rule).

The form of this vector is specified by the Graphics Program which displays the vector as a Gantt chart. The form requires that for each operation of a job (each bar of a Gantt chart) four pieces of data:

1. Machine number
2. Job number
3. Starting time of operation
4. Length of operation

This vector must be all integer. Thus, the user must translate the integer time unit to real time. Each time the simulation program on the 360/50 schedules an operation, the four pieces of data are recorded, and at the end of the run, the vector is sorted by ascending machine number, passed to a communications subroutine and then is transmitted to the PDP-9 and the Gantt chart is displayed.

4. Prespecified statistics about the schedule are calculated by the 360/50 program and passes as a vector to the PDP-9 and displayed with appropriate titles. A convention could be that element 1 of the statistics vector is mean flow time, element 2 is variance, element 3 is lateness, etc. Also, if the user has defined resources, the consumption by time period of each resource is stored.

This completes the first part of the simulation program's responsibility. The second responsibility is a result of the interaction of the user and the Gantt chart display. An interaction seniario could include:

1. The user wishes to try one of the built-in scheduling options. He uses the graphics light pen to send a command to the 360/50 computer to reschedule the shop using the specified rule. Thus, the 360/50 program has to start fresh on the job shop data and generate the new schedule, statistics, and resource consumption data.** Picture 1 (all pictures taken from the graphics tube showing actual runs. Pictures are in the back of the paper,) shows a Gantt chart with the menu of scheduling options.

2. After seeing schedules generated by the simulation program, the user may wish to decide on one to manipulate himself in order to improve it, or to introduce programmable constraints into the schedule. GJSSP gives the user several commands to enable him to move Gantt chart bars around to modify schedules and interject his own decision making into the scheduling process. Picture 2 shows the Gantt chart with manual changes. Upon completion of the user's modifications, a command is sent to the 360/50 to receive the Gantt chart vector. The simulation program has two responsibilities here. It must check the logic of the user's Gantt chart. Does the new schedule violate any of the pre-specified technological orderings? If a constraint has been violated, a dummy Gantt chart is sent back to the graphics computer and an error message is displayed. If no errors are encountered, the simulation program calculates the statistical values of the new schedule and transmits them to the graphics computer.

3. More complex shops include the consumption of scarce resources as a problem area. The amount of any resource consumed at time t is a function of the jobs that are on machines at time t. Different schedules give different resource consumption patterns. The problem of scarce resource consumption brings with it a new set of criteria such as minimizing the variance of utilization of the resource. A generated schedule that gives an acceptable Gantt chart may show a poor resource consumption chart. Different schedules are then tried with the job shop program generating schedule and resource vectors, computing statistics and checking for logic while the user views Gantt and resource charts and sends commands. After a suitable schedule is found, hard copies of all displays can be printed on a Calcomp or saved on DEC tape.

This scenario accurately describes a simple use of GJSSP. The rest of this paper will first describe the interesting aspects of the 360/50 simulation program, and how each of the three programming languages deal with each aspect. The remainder of the paper discusses additions that are required to the initial simulation program to give GJSSP more power. Thus, each of the programming implementations will be discussed in light of difficulties encountered in modification and extension of each implementation. The conclusion mentions some of the design goals and application areas of GJSSP.

INITIAL PROGRAM IMPLEMENTATION OF GJSSP

The following is a discussion of the problem areas in implementation of GJSSP and how each language treats the areas.

The important problem areas are:

1. World View of Language Applied to GJSSP. World View is defined by Gordon[5] as the set of concepts used for describing the system.

2. Timing mechanism (especially the problem of simultaneous events).

3. Re-run technique (ability to generate another schedule on command from graphics console and output to the graphics console).

4. Running time.

** Note that all specified results are presented to the user. Non-graphics systems typically schedule to optimize a weighted function. However, the power of interactive graphics allows the user to view statistics and graphs of criteria, rather than cover this data with weighted functions. GJSSP lets the user trade off in the n space of criteria values.

1. The world view of SIMSCRIPT II structures the job shop by defining machines and jobs as permanent entities. Jobs are defined as permanent for this application since they are not randomly generated. A specific shop with given machines and jobs is dealt within GJSSP. Since SIMSCRIPT II programs are structured in the PREAMBLE and since SIMSCRIPT II programs are self-documenting, the relevant characteristics of the implementation are shown by a few lines of code.

```
EVERY MACHINE OWNS A QUEUE AND HAS A STATUS
EVERY JOB OWNS A PERMANENT SEQUENCE AND BELONGS TO
A QUEUE AND HAS A DUE DATE AND A PRIORITY
TEMPORARY ENTITIES
EVERY OPERATION BELONGS TO A SEQUENCE AND HAS A
PROCESS TIME AND A ROUTE AND A RESOURCE
EVENT NOTICES INCLUDE END OF PROCESS
```

The structure of the SIMSCRIPT program is clear. Jobs own sets called SEQUENCE which contain OPERATIONS which in turn have attributes of ROUTE (machine number), PROCESS TIME and RESOURCE. The only event in the system is an end of process. Upon completion of an end of process, the job is sent to the queue of the machine denoted by the attributes of the next operation contained in the sequence of the job. Then the operation is removed and destroyed. When all operations in the sequence of the job are gone, (the set is empty) the job leaves the shop and statistics are generated. When a job goes to the queue of its next machine, a function is called to determine the job's priority of the job from conditions of the shop or the attributes of the job depending on which scheduling rule is used.

The significant advantage of SIMSCRIPT II's world view applied to this problem is that the program is coded, and in fact, looks like the above paragraph of verbal description. User defined structures like entities, attributes, sets and events require no forcing of the structure of the job shop. The PREAMBLE statements define and provide the structure of the remaining program.

The world view of GJSSP requires that the job shop fit the structure of the language. Transactions are defined as jobs, while facilities and queues are respectively machines and waiting lines for machines. The transaction jobs are GENERATED at time and move through the job shop in the following way:

```
ROUTE   ASSIGN      3,FN*1
        TRANSFER    FN,201
SHOP    QUEUE       P3
        SEIZE       P3
        ADVANCE     FN*5
        RELEASE     P3
```

As in the SIMSCRIPT II implementation, the shop structure is straightforward. Parameter 1 contains the job number of the transaction. FN*1 references the function that contains the sequences of machine numbers for that job. One function per job with a list of machine numbers with a zero at the end performs this task. The transfer block sends the transaction to a routine that assigns a PRIORITY which is used to rank the QUEUE. Upon SEIZING the machine, the job is ADVANCED by a time unit function determined by parameter 5 which is the job number plus 100 (if 100 jobs are to be provided for).

Test blocks are used to determine if the job goes to a zero value machine and if it does, statistics are recorded in save-values and the job leaves the shop. Due dates and resources are all contained in functions.

The job shop structure fits well into the world view of GPSS. Conceptually, jobs as transactions are easily viewed as particles flowing through a shop containing queues and facilities.

The world view of the FORTRAN implementation deals with several matrices and vectors that are used to structure the job shop. An event vector (one element for each machine) contains the absolute time of an end of process. The vector is searched for the minimum time and that element of the vector is the machine of the next event. A matrix with the rows machine numbers contains job numbers of jobs waiting for the machine. Each time a job arrives or leaves the machine, the machine row containing job numbers is shifted. Jobs are placed in the row by a sort routine that sorts according to the scheduling option used. Processing time is determined by a job number by machine number matrix and sequence by a job number by operation number matrix. Operation numbers are kept on a vector where each element i contains the number of operations completed for job i.

The program contains no list processing structure for the event or queue arrays. It is a pure logic test program. If you decide to program in FORTRAN, why try to spend the time with list processing sophistications. For example, this program uses bubble sorts which is easier to program than pointer and list sorts.

This implementation of the job shop contains no helpful self-documentation. Complex subscript nesting and multiple DO loops completely lose the job shop structure and thus make the model hard to rationally follow and to debug.

The touted abilities of high level simulation languages in presenting suitable world views to facilitate model building are apparent in comparison with FORTRAN. SIMSCRIPT II requires the model builder to be more creative than the user of GPSS's prestructured format, but offers no special difficulties in this application.

2. The timing mechanism for determining events is a problem area that caused the greatest implementation difficulty. The problem arises due to simultaneous events. For example, when a job leaves a machine, the next job should not be scheduled until other jobs that may be sent to that machine at the current clock time arrive. The FORTRAN program uses interval time advance. All jobs that finish processing at time X are sent to their next machines before time X+1. All timing information in GJSSP must be an integer since the graphics program is in terms of arbitrarily selected time units and references to partial times are not allowed. This means the simpler interval method of simulation can be chosen for the FORTRAN program.

A PRIORITY, BUFFER Block of GPSS is suitable to overcome this problem since all instructions in the current event chain are allowed to take place before queue sorting. SIMSCRIPT II requires a more complicated approach, but the language has the capabilities. The implementation looks ahead in the event list and tests if there are more events at the current clock time. If there are, control is returned from the event routine until all concurrent events have happened. Then scheduling of new events proceed. This example points out one of the significant powers of SIMSCRIPT II. The event list is accessible to the programmer and that leads to many possibilities for development of dynamic and look ahead scheduling rules.

3. Re-run technique and communications.

The implementation must have the capability of communicating with and doing several tasks on demand from the graphics console. Communication for the SIMSCRIPT II program assembles into a vector four pieces of data (job and machine number, start and process time which define a Gantt chart bar) each time a job leaves a machine. After all scheduling has taken place (the event list is empty) control goes to the line after STARTSIMULATION. The vectors of Gantt chart bar information are assembled and grouped by machine number and the communications routine is called. Upon returning from the graphics console, two possibilities occur. The user may just want a logic check on sequence and a statistical report, in which case, the vector of Gantt chart bar data is assembled by jobs by creating operations and filing them back into the sequence set. For each job, this sequence is checked against an initially stored sequence, and errors are reported. Statistics are compiled in all cases by scheduling an event routine.

The second task is a reschedule. In this case, program control goes to the beginning of the SIMSCRIPT II program and fills the sequence set from a permanent set containing operations. The number of the new scheduling option to be used is picked up and used for branching in the sorting function. The FORTRAN program accomplishes these tasks in much the same way. The operation vector is initialized to 1 and program control goes to the beginning of the main section. The jobs waiting for machine matrix is zeroed as is the clock and the simulation continues.

The GPSS implementation is the most cumbersome. Each job, when scheduled, updates a matrix savevalue that contains the relevant Gantt chart information. The last scheduled job (transaction) (other transactions that leave the shop are put into a user chain) goes through a HELP block that is used for communications. After communication, the transaction UNLINKS the other jobs; all operation number parameters are initialized to 1; and the jobs are sent to the shop structure routine. The problem with the implementation is the sorting and the logic of the unlinking transaction.

Since a transaction is used for testing and swapping a bubble sort over the communication savevalue matrix involves looping the transaction N x (N-1) times where N is equal to the number of Gantt chart bars. This is time consuming. Further, since the transaction represents a job to be scheduled if another run is desired, the identity of the job must be untouched and logic has to be programmed to insure that the job returns to be rescheduled again.

4. Running time is an important criteria for GJSSP. The system is designed interactively and response time is of major interest. Since the program runs in LCS, execution time can be slow. Consider a request to generate 200 schedules using random priorities, and display the schedule that gives the minimum mean flow time. Recall that such a request is initiated from the graphics console. High speed execution time (360/50) for the implementations is as follows:

FORTRAN	1.06 minutes
GPSS/360	6.22 minutes
SIMSCRIPT II plus	1.82 minutes

Actual response time (since the actual job is executed in low speed core) is about 4 times as long. Compiling time and core usage are not important for this application.

Comparison of the three implementations reveals that the Fortran program is a logic or engineering program. It shows nothing of the job shop structure. However, the simplification in timing and the ease of re-run made the initial programming as easy as the SIMSCRIPT II implementation. SIMSCRIPT shows the job shop structure very well, has competitive execution time, and has the best input/output procedures. GPSS was the easiest program, however, it has the worst input/output and execution time.

DISCUSSION OF EXTENTIONS TO THE IMPLEMENTATIONS

The initial implementation of GJSSP was in FORTRAN because the communication conventions are widely known. However, the FORTRAN program became unwieldy due to the complexity and the lack of rationality of the logic. Each time the program was looked at to modify it to include new scheduling rules or more extensive statistical reporting, the complexity of nested DO loops and subscripts made reprogramming difficult. This points out the problems involved in changing programs when the world view of the language doesn't fit the application. GPSS was removed from consideration because of its lack of power in performing some of the sophisticated scheduling rules, statistical reporting, and cumbersome input. For example, a branch and bound scheduling procedure will be implemented. Although there are no existing branch and bound or linear programs in SIMSCRIPT there is no reason why a powerful language like SIMSCRIPT cannot be used. Since unlike GPSS, SIMSCRIPT II is not only a simulation language, but also a programming language, the conclusion in selecting SIMSCRIPT is that the flexible world view it offers fits the job shop structure better than the structure of FORTRAN. Further, its full range of power makes it more desirable than GPSS.

While it is possible to give precise man hour times involved in certain reprogramming tasks, figures are always open to question. It is of more interest to examine a typical implementation of a new scheduling rule in SIMSCRIPT II. Consider some of the suggestions in "Job Shop Scheduling by Means of Simulation and an Optimum--Seeking Search"[2], Dr. Emery, University of Pennsylvania. Dr. Emery discusses a two stage priority determination rule to be used to assign jobs to a currently available machine. Stage one computes six data points on competing jobs and removes them from further consideration if they don't meet threshold or tolerance intervals. Jobs that pass all six screens get a stage two weighted priority function computed, and the job is selected that has the highest value.

Stage-One screening includes:

1. external priority class
2. C/T rule
3. time in current queue
4. work remaining/process time of current operation
5. processing time of current operation
6. size of queue at jobs next machine

Dr. Emery's paper gives a comprehensive discussion of the screening technique. A FORTRAN implementation would include vectors for each criterion, and presumably be somewhat hard to program, debug, and interpret if one had started with an existing program and desired to modify it to include Dr. Emery's work. A SIMSCRIPT II implementation would be easily done and self-documenting. Generally, a SIMSCRIPT II implementation for the GJSSP would take the following form.

1. external priority is read in initially as an attribute of the entity JOB.

2. C/T rule would be another attribute of JOB. It's value would be computed whenever an end of process event is reached (recall the logic of the program is that upon end of process the job just completed goes to its next machine queue, while the machine just finished selects the next job to be processed.

The C/T rule is defined as

$$\frac{\text{wait time} - (\text{due date} - \text{remaining processing time})}{\text{current processing time}}$$

All the terms except for wait time are available as attributes. Wait time is defined as the total processing for each of the jobs on queues of machines that the job goes to. This value could be found by the following SIMSCRIPT II statements:

```
FOR EACH OPERATION IN SEQUENCE ( JOB N)
FOR EACH JOB IN QUEUE (ROUTE OPERATION)
ADD PROCESS TIME (OPERATION) TO WAIT TIME (JOB N)
```

Wait time (job) would be used to calculate the C/T rule value which would be stored in another attribute of job.

3. Time in the current queue is found by recording Time V initially in an attribute, then at each end of process event, calculate the difference between the attribute and the current Time V.

4,5. Work remaining is found by looping over the remaining operations filed in the sequence set of the job, and summing each process time.

6. Work at the next queue is found in the same way as the wait time of the C/T rule.

The SIMSCRIPT II implementation would only allow the above attributes to be calculated when the user has specified Dr. Emery's procedure to be used to schedule. When his rule is used, the attributes of jobs on a machine queue are calculated and the job that meets all the thresholds and tolerances and has the highest weighted priority is removed from the queue set of the machine. Incidentally, the constants of thresholds, tolerances and weights can be entered directly from the graphics console. This would allow the user to do his own sensitivity analysis interactively. This extension of the implementation is very easy and clear in SIMSCRIPT II. Our FORTRAN version took much longer to program and debug, but man-hour programming time comparisons of languages is a risky business. It is sufficient to state that the additional programming in SIMSCRIPT II is easy because the flexible world view of the preamble statements and the concept of the language fits the structure of the job shop, whereas the additional FORTRAN programming to add Dr. Emery's work to the existing implementation was much harder because of the complexity of the logic. SIMSCRIPT offered no problem in handling the complex priority rule and other logic.

CONCLUSION

Other work[3,4] has shown the feasibility of graphics scheduling. GJSSP is quite different than these papers[3,4] (IBM 2250 Graphics) since the graphic commands and visuals are modified, and the simulator and scheduling package communications implementations use different methods. However, from other work, it is safe to assume the relevance of graphics (and thus, GJSSP) in the scheduling function. Since GJSSP is a research project, no specific application was in view. The implementation is general enough to include project scheduling with resource consumption, flow shops with parallel processing or other practical problems.

The object of this work is to introduce computer graphics scheduling and relate the characteristics of a programming language to serve as a job shop simulator. Since GJSSP is a powerful scheduling technique, and since it is constantly open to growth in terms of adding sophisticated scheduling rules, the implementation must be done in a powerful, flexible programming language.

REFERENCES

1. Conway, Maxwell, Miller, Theory of Scheduling, Addision-Wesley, 1967.

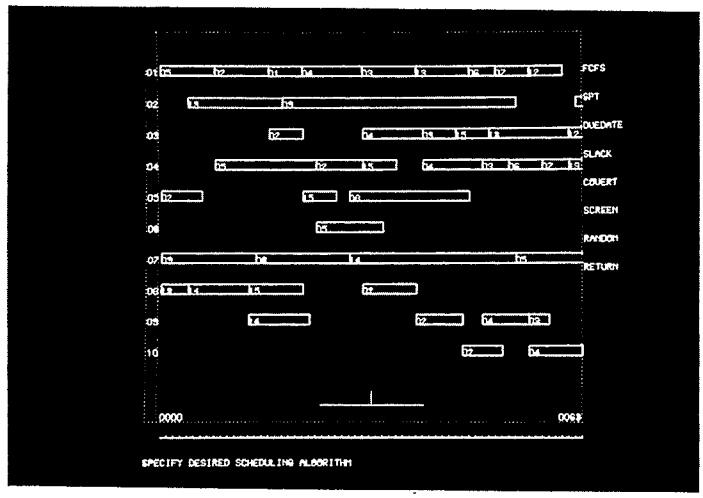
2. Emery, J. C., Job Shop Scheduling by Means of Simulation and an Optimum-seeking Search. Proceedings of the Third Conference on Applications of Simulation, 1969.

3. Jones, Hughes, Enguold, A Comparative Study of Management Decision-making from Computer-terminals. Proceedings, 1970 Spring Joint Computer Conference, AFIPS Press, 1970

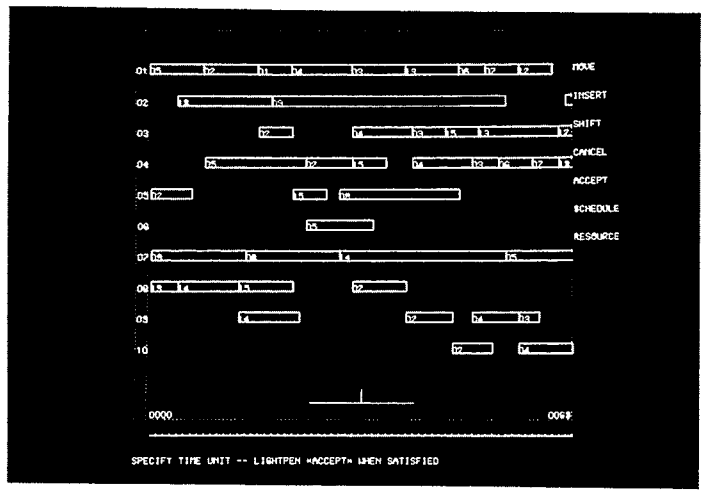
4. Bell, T. E., Graphical Analysis Procedures for Simulation and Scheduling, UCLA Ph.D Thesis, 1968.

5. Gordon, G. System Simulation Prentice-Hall, 1969.

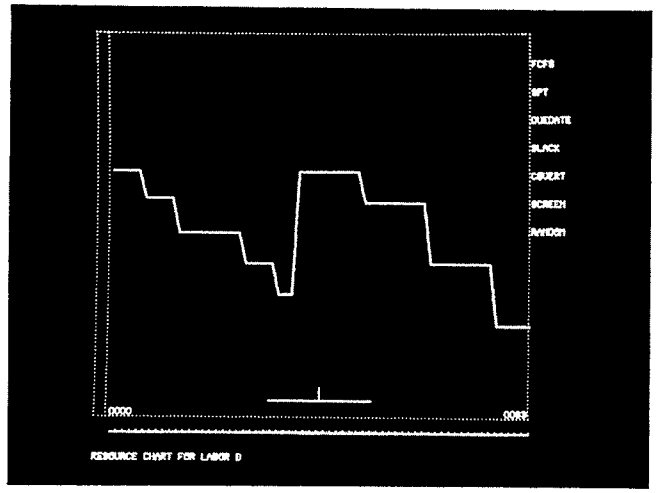
Acknowledgements to my colleagues working in Graphics, R. Sedgewick, J. McDonald.



PICTURE 1: Gantt Chart With Menu of Scheduling Options



PICTURE 2: Gantt Chart With Manual Change Commands



PICTURE 3: Resource Chart (Vertical Axis is Manpower Used)

All pictures taken from Graphics Tube from actual interactive runs. Machine numbers on vertical axis, horizontal axis is time.