MANAGEMENT SCIENCE
Vol. 13, No. 3, November, 1966
Printed in U.S.A.

HEURISTICS IN JOB SHOP SCHEDULING*†

WILLIAM S. GERE, JR.‡

Yale University

The problem is that of scheduling jobs with diverse routings on the productive facilities in a shop such that the respective due dates are met, or failing this, the sum of lateness times is minimized. The approach is simulative in that the operation of the shop is simulated in a Fortran program, but in addition to the straightforward use of priority rules for determining sequences of jobs on the machines, a number of heuristics or rules of thumb are incorporated. Both the static and dynamic problems are investigated. The efficacy of a small number of heuristics in combination with certain priority rules is demonstrated.

1. Summary

The problem is that of scheduling jobs with diverse routings on the productive facilities in a shop such that the respective due dates are met, or failing this, the sum of lateness times is minimized. The approach is simulative in that the operation of the shop is simulated in a Fortran program, but in addition to the straightforward use of priority rules for determining sequences of jobs on the machines, a number of heuristics or rules of thumb are incorporated. Both the static (all jobs on hand at time zero) and dynamic problems (new jobs admitted from time to time) are investigated. The efficacy of a small number of heuristics in combination with certain priority rules is demonstrated; in particular the "alternate operation" and "look ahead" routines will improve schedules significantly when used to augment a reasonable priority rule; the selection of the priority rule itself is a relatively unimportant matter. Further improvement will result when the schedule is re-run with enhancement of priorities for the late jobs.

The program also appears to yield highly satisfactory solutions for the "minimum make span" problem, a static problem in which the last job is to be completed as soon as possible.

2. Job Shops and Scheduling Problems

A job shop has been variously defined, usually in contrast with a "production" shop, in terms of the arrangement of equipment [15] for example, or the diversity of routing of the jobs passing through the shop [4]. We will say that "job shop" scheduling connotes a (large) number of jobs with diverse routings which will compete for time on common machine facilities, and the problem of lot or batch size is either solved or irrelevant.

* Received July 1964 and revised April 1966.

† Work on this project was supported in part by a Ford Foundation Doctoral Dissertation Fellowship, and a grant-in-aid from the Society of the Sigma Xi; use was made of the computing facilities at the M.I.T. and Yale University computation centers.

‡ Now at United States Rubber Company, Naugatuck, Connecticut.

What is the shop scheduling problem? There are a number of problems falling under this rubric, variously called scheduling, sequencing, and dispatching problems. The scheduling of production in a plant usually refers to the overall planning of the disposition of the facilities for a considerable period of time, such as a month or a year. Those responsible for scheduling will assign gross numbers of productive units (e.g., machine hours, man hours) to accomplish certain categories of work over some period of time. Such a schedule is necessary for the overall planning of machines and manpower, for ordering parts, tools, and supplies, and for budgeting of funds. In day-to-day scheduling, certain jobs or shop orders are released to the shop, and specified quantities of certain items are assigned to particular men and machines for accomplishment at a particular time. When a job is released to the shop, it has already been engineered and a due date has been established by which time the job should be completed, in order that it be shipped to the customer on time.

We will deal with the lower level, day-to-day scheduling problem in which each job has a due date; if a job is not completed by its due date, a cost penalty is incurred; we aim to minimize this cost via a good scheduling procedure.

Much of the research on this and similar problems has involved computer simulation [1, 2, 3, 11, 15] as a vehicle for testing various priority rules to be used in determining which job should have preference whenever there is a potential conflict on a machine. (These and other approaches to shop scheduling are described in references 5, 14, and 20.) In the work reported here, a version of the job shop scheduling problem which is not far removed from reality has been attacked heuristically, that is by the use of priority rules together with heuristics or rules of thumb in a computer program which will test the goodness of the various rules, alone or in combination, and will produce "good" results for a problem submitted for solution.

We use the word "heuristic" to mean "rule of thumb"—a meaning which has evolved in the past few years as heuristic programming [18, 19] has been applied to management problems [9, 10, 17, 21]. Kuehn and Hamburger [10] discuss current interpretations of the words "heuristic" and "algorithm".

3. Specification of the Problem

The output of a shop is a number of completed "jobs". A single job may involve only a small amount of work on one productive facility, or it may involve many hours of productive time on many facilities. *Job* refers to the work that is performed, and also the physical entity that is the object of the work. A job comprises one or more tasks or *operations*. We say that each operation is performed on a *machine*. The "machine" (the processing facility) may be a drill press, a paint spray booth, an ion exchanger, a work bench, a typewriter. The *routing* of a job is a list of the machines on which the job will be processed, in order. The job file is a listing of the routing, operation times, and due date for each job.

The scheduling problem with which we are concerned is the following:

A number of jobs, each comprising one or more operations to be performed in specified sequence on specified machines and requiring certain amounts of time, are to be scheduled such that all due dates will be met, or failing this, costs due to lateness will be minimized.

In the static scheduling problem the job file is given. In the dynamic problem there is an initial set of jobs on hand, and more jobs appear from time to time. The discussions in the paper are concerned with both the static and dynamic problems, the former because analytical formulations of the problem are more nearly tractable, the latter because this is the usual problem in a shop. The computer program is designed to deal with both the static and dynamic problems; an input datum will summon one program or the other. Since the make-span problem is a particular type of static due-date problem, the program can be used in testing procedures for reducing the make-span.

We attach several conditions to our problem, as follows:[1]

1. No machine may process more than one operation at a time.
2. Each operation, once started, must be performed to completion (no preemptive priorities).
3. Each job, once started, must be performed to completion (no order cancellations).
4. Each job is an entity; that is, even though the job represents a lot of individual parts, no lot may be processed by more than one machine at a time. This condition rules out assembly operations.
5. A known, finite time is required to perform each operation[2] and each operation must be completed before any operation which it must precede can begin (no "lap-phasing"). The operation time includes set up time.
6. The time intervals for processing are independent of the order in which the operations are performed. (In particular, set up times are sequence-independent and transportation time between machines is negligible.)
7. In-process inventory is allowable.
8. Machines never break down and manpower of uniform ability is always available.
9. Due dates are known and fixed.
10. The job routing is given and no alternative routings are permitted.

When a job is not completed by the due date certain costs are incurred. These costs include: (1) direct dealings with the customer—paper work, telephone calls, executive time taken up, (2) penalty clause in the contract, if there is one, (3) loss of good will resulting in an increased probability of losing the customer for some or all of his future jobs, or perhaps in a damaged reputation which will turn other customers away, and (4) expediting (the job is moved quickly through the shop at the possible cost of extra set-ups, double handling of materials, inefficient use of workmen and machinery). The aggregate of the several costs is monotonic increasing with time of lateness. It appears reasonable to presume a linear cost function for lateness and yet one might just as readily rationalize that a quadratic function more nearly fits the real situation.[3] Our computer program calculates

[1] Some of these conditions are taken from [20], p. 298.
[2] See [4] and [13] for some justification for this.
[3] See [8] and [12] for example, for discussions of lateness costs.

both linear and quadratic cost functions for lateness—and for earliness also—so the scheduling results may be analyzed in any of several ways. We have used the sum of tardiness hours as our criterion function.

4. Definitions, Priority Rules, Heuristics, Conjectures

In order to lay out a schedule we need a rule or set of rules. Since the scheduling problem is one of determining precedence or ordering in time among a number of jobs, an operational scheduling rule includes a means of determining priorities of jobs. The scheduling rule (or complex of rules) may do more than this; for example, it may say: "Schedule the job with the least slack, but if this forces another job to be late then schedule that other job instead."

Definition: A *priority rule* or priority function is that function which assigns to each waiting job a scalar value, the minimum[4] of which, among jobs waiting at a machine, determines the job to be selected over all others for scheduling. In the case of a tie, the job with smaller job number is selected.

Thus, the job slack priority rule would assign a number (number of slack hours now remaining) to each job waiting to be processed on the machines.

Definition: A *scheduling rule* dictates which job among those waiting for service is to be scheduled in preference to the others. "Scheduling a job" means scheduling the next operation of the job.

By these definitions, priority rules comprise a proper subset of scheduling rules. A scheduling rule may include one or more heuristics in addition to or instead of a priority rule.

It is reasonable that the rule relate explicitly to the data at hand, that it involve job routings, operation times, machine loads and so on, and that some other rule which ignores the data at hand, such as a random rule, will not schedule very well. The data at hand for a scheduling problem are contained in the job file or can be directly derived from the job file.

We say that a scheduling rule is a function of the job file when the decision made (choice of job) by the use of the rule is dependent on at least one of the following: Number of operations, operation times, due dates. For example, "job slack" is a function of the job file, it equals the due date measured in time units from the present minus the sum of operation times. Machine utilization is also a function of the job file if measured in terms of operation times and due dates. A "first come, first served" (FIFO) rule is not a function of the job file,[5] nor is a priority equal to the job number, since job numbers are arbitrary; such rules are in effect random rules.

Definition: The class of priority rules we shall call "random" includes any rule whose priority function is not a function of the job file.

[4] For all rules tested herein (except for the "insert" rule), minimum priority value means "top" priority. Of course, ratings could be devised for which the maximum value represents the greatest urgency.

[5] Although the possibility of a job appearing in a queue is, of course, dependent on the job file, the priority of a job is determined solely from the time the job appeared at the machine.

之後可以當作baseline來用

Since the "first come, first served" rule is included in the "random" class we would expect it to be no more effective in meeting due dates than a purely random priority rule. We would draw the same conclusion for an arbitrary rule such as: Priority rating equals job number.

We measure "effectiveness" of a scheduling rule by the amount of tardiness of job completions, or more generally, the cost of not meeting due dates exactly, resulting when the rule is applied. This criterion function may or may not include cost of job completion ahead of the due date; it may be of simple form, such as "sum of late hours". "Criterion function" as used below refers to the cost of not meeting due dates exactly.

Definition: The *effectiveness* of a scheduling rule is measured by the value of the criterion function which results when the rule is followed.

Effectiveness may be measured with respect to a given job file or a class of job files.

Job priorities may be set before scheduling begins and then either remain unchanged or vary in some way independent of the schedule and of the job file. The priorities in such a case are static. We would expect an effective priority rule to be dynamic, reflecting the status of jobs from time to time as the schedule progresses. All of the non-random rules discussed in this paper are dynamic; priorities must be updated regularly. Examples of static priority rules are: Priority equals due date; priority equals reciprocal of number of operations. With these rules jobs are ordered by priority before scheduling begins, and the order remains unchanged thereafter. If a static priority rule is devised which appears to be a "good" rule, then there should be a better rule which is the dynamic analogue of that static rule; for example, the dynamic analogue of the due date rule is job slack; in the former, priority equals due date minus starting time, in the latter, priority equals the due date minus present time minus processing hours remaining. We contend, without experimental verification, that there is a dynamic rule which is more effective than a given static rule; the dynamic rule takes into account additional information not available to the static rule.

Most priority rules are intended for use with all job files. When several rules are tested with a job shop simulator the "best" rule is sought, the one which will be most effective most of the time, regardless of the particular job files. On the other hand each particular job file could be handled as a separate problem. One might take a "jig saw puzzle" approach and lay out a Gantt chart of the schedule by trial and error. In this event it is unlikely that priority rules will be made explicit; however, an intuitive sense of the job slack for example will underlie some of the decisions. We will say that the "general rule" approach is less costly to operate but the schedules are poorer, as compared with the "specific problem" approach. If scheduling is to be done by computer, the specific problem approach will be very costly since it is a formidable task to program the many heuristics which a human will employ in this moderately complex task, and furthermore the running time might be appreciable. Also, the specific problem approach may not be worthwhile in the dynamic case, since future jobs will necessitate changes in the schedule as now laid out.

We propose the following as a reasonable intermediate approach: Make use of "good" priority rules but tailor them to the particular problem at hand. For example, use a job slack rule to determine priorities but if following the rule causes a job of lesser priority to be made late, then the rule should be broken. Thus we combine one or more priority rules with certain heuristics. What *priority rules* are used? Those which show themselves to be effective, in tests run on a shop simulator. Since there are few published results on the relative merit of various priority rules when the problem is to *meet due dates*, a priority-rule testing program was devised as a part of this research. What *heuristics* are used? Heuristics which seem reasonable, and worthy of being tested in the program, may be devised introspectively by taking note of the rules intuitively called into play as one attempts to lay out a schedule, and also by asking others to record their ideas.[6] Once a repertoire of priority rules and heuristics is at hand, schedules can be laid out and various combinations of priority rules and heuristics tested. Furthermore, the heuristics should include an answer to the question: If the schedule isn't good enough—or if we think we can do better—what should be done?

A point to consider is this: Once the job file information is at hand all of the information necessary to tell how good a rule will be is available; to determine how good a rule is, one might simply proceed by trial and error and lay out the schedule (that is, simulate the shop operation) but one should have a fair idea, a priori, of the outcome by making an analysis of the job file data.[7] Thus, the testing program includes a job file analysis; results of the test runs are examined with respect to the analysis with the intent of finding significant relationships. Then in the heuristic program[8] the job file analysis directs the program toward the priority rule or rules found to be most effective in previous runs.

What is being sought when researchers analyze queues, compare priority rules, and so on? In effect, what is desired is a perfect prediction of delay (wait) time, that is, a knowledge of job completion time, together with a rule which will minimize or maximize some function of completion times. Our aim is to minimize costs of not meeting due dates; ideally we want each job to be completed just at its due date. Furthermore, we want jobs completed "as soon as possible:" that is we want to be able to set the due dates a minimum time into the future and still be able to meet them. We desire both predictability (because we would like to set due dates correctly) and control (because we want to meet the due dates).

What priority rules should be effective? It is reasonable to expect that effective priority rules will take into account due dates or processing times or both. Furthermore, since each operation is another opportunity for delay as the job waits in a queue at the machine, the number of operations appears relevant. Also, the

[6] See [19] for a discussion of the role of protocols in devising "intelligent" computer programs.

[7] If the data analysis is more costly than a number of simulation runs, then the analysis may lose its appeal; the question is one of economics, not methodology.

[8] The program at present does not accomplish this; it is still being used as a testing program for the priority rules and heuristics. As will be seen later, initial trials give little encouragement for using the job file analysis to select the "best" priority rule.

machine loading should be a significant factor; expected delays are greater on the heavily loaded machines. In consideration of the above, and of results obtained by other researchers, [2, 4, 15] we have included the following priority rules in the testing program:

1. Job slack
2. Job slack per operation
3. Job slack ratio (job slack hours divided by hours remaining until the due date)
4. "Modified" job slack ratio, which takes machine loading into account by adding to each operation time the expected delay time associated with it.
5. Length of next operation. This is the "SIO" rule (shortest imminent operation).
6. Length of next operation together with job slack ratio under certain circumstances.
7. First come, first served.
8. Random

The last two rules are employed to serve as bench marks and to test the hypothesis that a rule which does not depend on the job file, such as first come—first served, is equivalent to a random rule. See Appendix B for a formal definition of rules 1 through 6.

We now set forth a number of conjectures, to be tested presently:

Conjecture 1: A scheduling rule whose priority function is not a function of the job file is no more effective than a purely random rule.

Conjecture 2: Each of rules 1 through 6 is more effective than either rule 7 or rule 8.

Conjecture 3: (a) If the several jobs have different numbers of operations, rule 2 (slack per operation) is more effective than rule 1 (slack).
(b) Otherwise, rule 2 is not less effective than rule 1.

Conjecture 4: (a) If the jobs have different due dates, rule 3 (slack ratio) is more effective than rules 1 and 2.
(b) If jobs have indentical due dates, rule 3 is not less effective than rule 1.

Conjecture 5: (a) If machine loadings vary appreciably, say there is some machine twice as heavily loaded as another, rule 4 (modified job slack ratio) is more effective than rules 1, 2, and 3.
(b) If machine loadings are approximately equal, rule 4 is no less effective than rules 1 and 3.

The SIO rule has been shown to be superior in getting jobs completed as soon as possible (as cited above) but when due dates are imposed the jobs with lengthy operation times may be quite late and in the dynamic problem, a job may be delayed a very long time. We see no impelling reason for discriminating between this and the other rules, but conjecture thus:

Conjecture 6: (a) Rule 5 (SIO) is not less effective than rule 1 (slack) for the static problem.
(b) Rule 5 is less effective than rule 1 for the dynamic problem.

Conjecture 7: Rule 6 (SIO—slack ratio) is more effective than rules 1 (slack) and 5 (SIO).

In summary, we expect the rules to rank approximately as follows in general, in decreasing order of effectiveness:

i. Rule 4 (modified slack ratio), rule 6 (SIO and slack ratio).
ii. Rule 2 (slack per operation), rule 3 (slack ratio).
iii. Rule 1 (slack), rule 5 (SIO).
iv. Rule 7 (first come, first served), rule 8 (random).

The comments and hypotheses above refer to the "blind" use of specific rules which are inviolable. Granted that the rules are dynamic, always reflecting current conditions, and that they (rules 1–6) do reflect the content of the job files, still it seems apparent that even a good rule should be violated under certain conditions; in other words another rule should be superimposed upon the first one. Rule 6 is an instance of this: In the SIO-slack ratio rule, the job with the shortest imminent operation is selected from among those jobs which are somewhat urgent, according to job slack ratio; and if a job is doomed to be late it is scheduled, without regard to either rule. Thus we are moving away from the "general rule" approach toward the "tailor-made" approach; if the immediate situation calls for action extraneous to the priority rule, then take exception to the rule. This is just what is proposed in the description of several heuristics, which follows:

1. Alternate Operation

Continuing the line of reasoning above we might say: Schedule the operation according to the rule. Now check to see if this makes another job "critical" (that is, see if the slack of any other job has just become negative, or if positive, has reached a certain critical level). If so, revoke the last operation, and schedule the next operation on the critical job. Check again for lateness. If scheduling the second operation does not cause any job to be critical, whereas the first one did, then schedule the second one; otherwise schedule the first one (that is, the one which was dictated by the rule).

The heuristic just described is used as a supplement to the basic priority rule; we call this the "alternate operation" heuristic.

2. Look Ahead

When we use job slack or some adaptation of it as a priority rule, we are looking ahead to a limited extent imagining that the schedule is laid out before us; if job slack is positive for a job then that job will be completed on time unless it is unduly delayed because of conflict, that is, queuing, at the machines. If we could look ahead and anticipate every conceivable conflict then we could schedule optimally, but the apperception of all conflicts would entail near-exhaustive iteration of possible schedules. We can however look ahead a short ways at modest cost; when an operation is scheduled we may ask: Is there a critical (i.e. late or nearly late) job due to reach this machine at some future hour, yet before the scheduled operation is completed? If so, schedule that job. Check to see the effect of this on other jobs. Either let the schedule remain, or replace the operation with the previously scheduled operation, depending on the resulting job lateness effected. This is called the "look ahead" heuristic.

3. Insert

Once a "look ahead" job has been scheduled there is a period of idle time on the machine, starting at the "present" time. If there is a job in the queue whose next operation can be completed by the time the look ahead job is due to arrive at this machine, then the operation should be scheduled. The "insert" routine says that the longest operation which can be fitted into the idle time gap, should be scheduled. It is obvious that the insert heuristic is helpful (in contrast with its absence); there is a question whether it will occur often enough to warrant its inclusion in the program.

4. Time-transcending Schedule

If looking ahead is a key to effective schedules, then why confine the scheduling process to a movement through time? It might be well to schedule in this fashion: Determine priority rating for each job. Schedule the next operation of the job with top priority. Re-evaluate priorities, and repeat, always scheduling the next operation of the job with top priority. This procedure permits scheduling an operation at say hour 20, followed by another operation at hour 10, and so on. In this way the most critical job is the one being scheduled; it is conceivable that a "tight" job be scheduled to completion before another job has begun. There is no guarantee of success with this method; even with its time-transcending feature, it cannot anticipate future conflicts except by iteration—by laying out "pseudo-schedules" after an operation is scheduled to see that it is still feasible to complete all jobs in time. And if it is not feasible, it is not clear that the last operation scheduled should be changed; in fact the damaging conflict may have been set up by some other operation scheduled previously. We conclude without experimental validation, that a time-progression program with a look-ahead feature is as effective as a time-transcending program—and much easier to program for the computer.

5. Subset of Critical Jobs

While still considering the time-transcending approach we may propose that schedules be laid out by *jobs* rather than *operations*. If a job is "tight", schedule it in its entirety, then schedule another job and so on. Since this approach is inflexible and capable of spawning serious conflicts, we reject it out of hand, and then propose an intermediate approach: Select a subset of "critical" jobs; schedule these jobs according to a priority rule or set of rules; then schedule the remaining jobs around them. This approach too is inflexible.[9] One difficulty that might arise is the following: The critical jobs may utilize a machine for many consecutive hours; a non-critical job is forced to wait many hours at the machine, thereby

[9] Unless the "subset" schedule can be tampered with. See Heuristic 8.

becoming first critical and then late; if a critical job had been postponed a short time to permit scheduling the other job, then both would have been completed on time.

This discussion brings out an important point: A job which is classified as critical ex ante based on the job file analysis and/or the priority rule will not necessarily remain more critical than most other jobs; the fact that it is critical will give it high priority and hence its relative priority rating will tend toward the median rating as the schedule progresses; conversely an initially "loose" job is pushed aside and tends to become tight. Most dynamic priority rules are priority equalizers.[10]

Returning to the "schedule critical subset of jobs first" heuristic, we contend that at least as desirable results can be attained by scheduling all jobs together; as the schedule progresses one could project a pseudo-schedule for the remaining operations of the critical jobs, but the relative priorities of all jobs are continually accounted for anyway, and a truly critical job will receive the preference it deserves.

One promising feature of the "critical subset" heuristic is that it points out explicitly the conflicts within the subset. It is possible to resolve such conflicts by rescheduling the critical jobs before the remaining jobs are considered. But even then the difficulty cited above—conflict with otherwise loose jobs—is extant. In summary, the "critical subset" heuristic apparently offers no additional desirable features unattainable with a time-progression schedule with a look-ahead routine.

6. *Re-do with Adjusted Due Dates*

The notion of rescheduling the critical jobs leads to another, rather obvious, heuristic. After a schedule is completed, if it is unsatisfactory by some criterion, or if the scheduler thinks that improvement can be achieved, make out a new schedule, based upon another scheduling rule or upon the same rule with modified priorities of jobs. As for the first notion, the computer program does permit replication of schedules using other rules; indeed the testing program is set up for that very purpose. Concerning the second, the modification of priorities: If a job is late it is reasonable that it be given a boost in priority to help it along on the next run. Hence this heuristic: When a schedule is completed and at least one job is late, decrease the due date of each late job (for priority calculations) by the hours[11] the job was late; then lay out the schedule again. For the second schedule, the rule[12] will "tighten up" the jobs previously late, hopefully just enough to complete them on time in the revised schedule. This process of re-doing the schedule could be repeated, but it is likely that a degenerate, rob-Peter-to-pay-Paul situation will arise after the second schedule, therefore in the computer

[10] This is borne out by computer results. Frequently a job with a low (tight) priority function at the start will be completed in advance of its due date while another with initial high (loose) rating will be late.

[11] In the program at present, the due date is decreased by one-half of the lateness hours.

[12] We are referring to rules which take due dates explicitly into account, such as rules 1, 2, 3, 4, and 6 of the eight rules discussed previously.

program no additional schedules are attempted with the given scheduling rule. The "re-do" heuristic has intuitive appeal.

7. Flexibility

As a Gantt chart is laid out, now and then an operation will not quite fit between two other operations on a machine and a lengthy delay will result; if the operation could be "squeezed into" the available time, this job would be helped considerably and other jobs would not be delayed very much. It would appear that such a move would usually improve the schedule, providing the squeeze-in operation does not postpone very many operations on that machine. We call this "flexible" scheduling since we are violating the rigid time constraints. When the schedule is completed then the squeeze-in operations will be allowed to consume their expected processing times and the remainder of the schedule will be adjusted (lengthened) accordingly.

The desirability of employing this heuristic depends upon the manner in which the schedule is devised: If a time-transcending mode is employed, or if a subset of jobs is scheduled first, then opportunities for squeezing in do appear, but if a time-progression mode is used the only chance for placing an operation between two others already scheduled on a machine will be as an *insert* following a *look ahead* (described above). We conclude that if time-progression scheduling is employed without subset scheduling—and this is what we recommend, above—then there is no need for flexible scheduling. We believe that flexible scheduling merits further investigation.

8. Manipulation

A final heuristic may be analyzed with reference to the Gantt chart. Suppose that the schedule has been laid out and some improvement is desired; we could adjust priorities and run through the schedule again as discussed in heuristic 6. But if we want to shift to the "tailor-made" approach, attacking this job file as a unique problem instead of following the rules the second time, we might proceed to rearrange operations on the Gantt chart until a better schedule is attained. This again is the "jig saw puzzle" approach, or what we shall call the "manipulation" heuristic. Taken as a general approach, this encompasses a host of possible heuristics such as: "If you see a gap on a machine, try to move a job up into it, especially a late job"; "If there is a string of operations on a machine without any gaps try to move up the first operation in the string and see if that won't loosen up the others too"; "If there is a string of jobs try interchanging the first two or three jobs and see what happens"; "If there is a long wait before the first job gets onto a machine see if you can get that job or another one to the machine sooner—especially if the machine is a busy one". To program many of these heuristics is a great task, but one which should yield better schedules. Manipulation is not included in our heuristic scheduling program but is recommended as a future step in this research.

Now we offer some conjectures relative to the heuristics.

Conjecture 8: The alternate operation heuristic is effective. (It yields schedules significantly better than those resulting from the same situation without the heuristic).

Conjecture 9: The look ahead heuristic is effective.

Conjecture 10: The re-do heuristic is effective.

The last conjecture is true by definition, since the schedule selected is always the more effective of the two. The only question is whether the better of the two schedules is significantly better, in a practical sense, than the first schedule.

Furthermore since the alternate operation and look ahead routines operate independently:

Conjecture 11: The alternate operation in ally with the look ahead heuristic is more effective than either one alone.

In section 6 we report on initial tests of the eleven conjectures.

5. The Scheduling Program in Operation

As a job file is about to be scheduled certain data must be read into the computer. The classification of the job file and of the program features which are to be used is contained in a set of 16 separate digits which comprise the "KC" array of variables. The classification includes: Data generation (read in the job file, or generate the routings and operation times at random), machine groups (all machines are unique, or groups are permitted), job routing (jobs may or may not return to the same machine again), admission of jobs (static or dynamic), time horizon (look ahead routine is or is not used), and so on. Data input also includes several shop parameters: Number of jobs, operations per job, and machines; and several constants including the "critical" level of job slack, a constant for calculating the number of jobs to be admitted per day, and the upper and lower limits of factors for determining due dates. For the complete classification of the job file and scheduling strategy see Appendix A.

The job file contains specific information about each job. The jobs are numbered consecutively from one; machines are designated the same way. For each job there is the following information: (1) routing, that is, the machines to which the job must go, in order; (2) operation times on the respective machines; (3) due date. None, some, or all of this information may be read in on data cards; that which is not read in is generated.

In the Data Generation routine, random numbers are used to determine:

1. Machine Groups. If machine grouping is permitted, each machine is assigned to a group of one or more machines, not exceeding in number the limit set in the schedule classification ("KC" number).
2. Job Routing. In accord with the routing restrictions set by the classifications, job operations are assigned to machine groups, perhaps with a bias towards some machines.
3. Operation Times. Times are distributed from 1 to 10, rectangularly if a bias is not introduced.
4. Due Dates. The due date equals the sum of operation times for the job multiplied by a factor which is selected at random between the upper and lower limits specified as program input. The limiting factors might be 2.0 and 4.0

for example; then the time permitted for a job to be completed will be two to four times the actual processing time for the job.

5. Number of jobs admitted each day (for the dynamic program). The first few jobs are admitted on day one until a reasonable initial backlog is attained. The rate of admission of jobs on the other days is calculated from the shop parameters and this rate is used as the mean of a Poisson distribution from which observations are taken at random to determine the number of jobs admitted on each particular day. Total shop backlog is recorded daily during the scheduling process. While the first schedule is being run, if the backlog exceeds a specified limit, no jobs are admitted on the next day and all subsequent jobs are postponed one day.[13]

Once the job file has been established, an analysis of the job file is made. The analysis determines: Total job hours, total machine group hours by group, total number of operations, mean operation time, mean number of operations per machine and hours per machine, range of job slack and job slack ratio, ratio of minimum to maximum job slack and job slack ratio, and the ratio of lightest machine load to heaviest machine load.

In the Scheduling routine the priority rule is selected and scheduling begins. The schedule proceeds in this way:

1. Look at the first machine. If it is busy move to the next machine. If it is idle, check to see if there is a job waiting for the machine. If so, schedule it. If there is more than one job waiting, select one of them according to the rule.
2. Consider the next machine. Follow the same procedure.
3. Repeat step 2 until all machines have been considered.
4. Advance the "clock" one hour and repeat steps 1, 2, 3.
5. When all jobs are fully scheduled, go to the Evaluation routine.

The routines described below, Operation Selection, Alternate Operation, Look Ahead, Insert, First Come-First Serve and Random, and Evaluation are all linked to the Scheduling routine which serves as an executive for the scheduling process.

The Scheduling routine also handles the following duties:

1. Calculation of "modified" operation times for the modified job slack ratio rule. This entails determination of expected delay times based upon machine backlogs.
2. Scheduling the operations, by recording them in the "Gantt chart", a three-dimensional array which shows job number, start time, and finish time, for each operation, listed in sequence opposite the appropriate machine.
3. Housekeeping at the operation level (after each operation scheduled) and at the schedule level.
4. Daily housekeeping for the dynamic program. This includes a check on the jobs now in the shop and the backlog in hours for each machine group.

[13] Although this procedure is reasonable as far as shop operation is concerned (since in fact admission of new jobs will be postponed, or subcontracted, or put on a night shift, etc., as the shop becomes overloaded), the extent of job postponement is dependent on the first priority rule used. We would expect to obtain comparable results with the other rules, however, and do not feel that this is a serious issue.

The Scheduling routine controls not only the routines to be used for scheduling an operation, but also the move to another priority rule (scheduling the same job file again) or to a new job file.

The non-random rules (rules 1–6) which are formalized in Appendix B are used in the Operation Selection routine to choose the job which, so far, is the one to be scheduled. The job selected is called the "standby job". It will be the job scheduled unless one of the heuristics, e.g., alternate operation, replaces it with another job. In operation selection each job is examined in turn. If the job is "idle" and its next operation is to be performed on the machine under consideration, then it is in the queue waiting for the machine. If there are no "queue" jobs, move on to the next machine. If there is just one such job, it is the standby job. If there is more than one such job, the one with minimum priority rating is the standby job.

If there are at least two queue jobs, then the Alternate Operation routine is entered. Priority rating is calculated for each queue job (other than the standby job) as if the standby job had been scheduled. That job with minimum priority rating is the "alternate standby job". Between the standby and alternate standby jobs, the one with smaller priority rating, where priority is based upon the other job having been scheduled, is designated the standby job.

Next a re-examination is made (the Alternate Operation Check routine) to see if particular conditions exist which would make the above choice incorrect. The conditions are the following: Both jobs are rated late (negative priority rating as calculated above), at least one job is on its last operation, and the later job has an adjusted due date.[14] Under these circumstances the relative priority ratings may not be a true reflection of job urgency. The Check routine then re-evaluates the two priorities as above except that true due dates are used and, for modified job slack ratio, actual processing times are used, exclusive of expected delay time. The job with smaller rating then becomes the standby job.

If a job is selected for scheduling by either the Operation Selection or Alternate Operation routine, and that job is indicated to be late (priority rating is negative) then the job is scheduled. If it can still meet its due date, then the Look Ahead routine begins.[15] This routine determines whether there is a job now being processed on another machine, which will go to this machine next, and will arrive while the standby job is still being processed on the machine, and which has a priority rating of late or critical. If there is more than one such job, the one with smallest priority rating is chosen as the "look ahead job." Then the priority rating of the standby job is recalculated as if it were scheduled after the look ahead job. The look ahead job is scheduled if the following two conditions are satisfied: The new priority rating of the standby job is positive, and the priority rating of the look ahead job is no greater than one-half that of the standby job. The conditions for a look ahead job are stringent because there may be several queue jobs delayed by the look ahead job for several hours in excess of the processing time alone.

[14] The adjusted due date means that the schedule is being run off a second time and that this job was late in the first schedule.

[15] The Look Ahead routine is not used with rule 5, shortest imminent operation.

Once the look ahead job is scheduled there is a period of idle time on the machine just prior to the look ahead job. The Insert routine serves the function of filling the gap, by checking to see if there is at least one queue job whose next operation will fit in the gap, and, if so, by selecting the longest such operation for scheduling as the "insert job."[16]

When a schedule is completed, certain calculations are made in the Evaluation routine: Total earliness and lateness, and the linear and quadratic cost functions, numbers of jobs early and late, machine utilization, schedule make-span, mean queue length. Then the schedule is printed out as well as the status of each job. Control returns to the Scheduling routine where a new priority rule is selected, or the schedule is run off again with the same basic rules, or a new job file is obtained.

The Scheduling routine also will direct the first come–first serve and random scheduling; the entire schedule is laid out within one subroutine (there is no Operation Selection, Look Ahead, etc.) and then Evaluation takes place.

The program continues to accept new job files as long as there are data cards specifying the shop parameters and the classification of each new job file.

The program was written in FORTRAN for the IBM 709 computer.

6. Tests of the Conjectures

For testing the conjectures a number of job files were selected, with 4 to 6 machines, 6 to 60 jobs, 1 to 16 operations per job. Job times were generated rectangularly, with 1 to 10 hours per operation. (Detailed information on the job files, output, etc. is available on request).

Twenty five static job files were run, for all eight rules with each combination (presence or absence) of three heuristics—alternate operation, look ahead, and re-do. Sixteen dynamic job files were run for all eight rules, first without any heuristics and second with both the alternate operation and look ahead heuristics. In the schedules not more than one-half of the jobs were late for most priority rules. (If most of the jobs were late, the job file was not accepted for the purpose of testing the priority rules and heuristics.) The criterion was the sum of tardiness hours. Since the results for the various job files should not be treated as observations from a single population, or from populations with known parameters, distribution-free tests were made. Wilcoxen's matched-pairs signed-rank test was used. (See [16].) The results which are summarized below apply to both static and dynamic schedules unless noted otherwise.

Conjecture 1: Rule 7 (first come, first served) vs. rule 8 (random)
Results were not conclusive; rule 7 performed better than rule 8, but the difference was not significant.

Conjecture 2: As an extreme case, the poorest of rules 1–6 (rule 5) wes tested against rule 7.
Rule 5 was significantly more effective than rule 7. The conjecture is supported.

[16] Before the longest operation is selected a check is made to see if any job has a negative priority rating; if so, the latest such job is scheduled. This event is possible but unlikely.

Conjecture 3a: Rule 2 vs. rule 1 when there are different numbers of operations in the jobs.
Rule 1 was somewhat better than rule 2, contrary to the conjecture, but the sample size was small ($N = 6$ for static scheduling; test was not conducted for dynamic scheduling).

Conjecture 3b: Rule 2 vs. rule 1 when each job has the same number of operations. There was no significant difference in performance. The conjecture is supported.

Conjecture 4a: Rule 3 vs. rule 1
Rule 1 performed somewhat better than rule 3. The conjecture is refuted; apparently rule 3 is no more effective than rule 1 (or rule 2), and there is doubt as to whether it is as effective as rule 1 or rule 2.

Conjecture 4b: Not tested.

Conjecture 5a: Rule 4 vs. rule 1, for schedules in which machine loading is unbalanced.
Rule 1 performed somewhat better than rule 4. The conjecture is refuted; rule 4 is apparently no more effective than rule 1 (or rule 2), and there is doubt as to whether it is as good as rule 1 (or rule 2).

Conjecture 5b: Rule 4 vs. rule 1 for schedules in which machine loading is balanced. Results were inconclusive; rule 1 appeared to be somewhat better than rule 4.

Conjecture 6a: Rule 5 vs. rule 1, static schedules. The conjecture was refuted; rule 1 was significantly more effective than rule 5.

Conjecture 6b: Rule 5 vs. rule 1, dynamic schedules. The conjecture was supported; again rule 1 was significantly more effective than rule 5.

Conjecture 7: Rule 6 vs. rule 5; rule 6 vs. rule 1. The conjecture was supported in that rule 6 was significantly more effective than rule 5, and was refuted in that rule 1 appeared to be superior to rule 6.

Conjecture 8: Alternate operation heuristic. Tested for static schedules only. The use of the alternate operation heuristic was examined with respect to rules 1–6. The total tardiness time for all six rules was the criterion. For only one job file in 25 was the alternate operation routine detrimental and then only to the extent of one hour on the average for the six rules. (Comparison is based upon look ahead not being used at all.) The conjecture is confirmed very strongly; the alternate operation heuristic is effective.

Conjecture 9: Look ahead heuristic. Tested for static schedules only. The use of the look ahead heuristic was examined with respect to rules 1 through 4 and 6. (The look ahead routine was not programmed for rule 5.) The total tardiness time for these five rules was the criterion. In 25 job files, the tardiness was reduced by the use of the look ahead routine in 15, there was no change in 7 others (look ahead was not actually put into effect in these files), and in the remaining 3 files the average loss was less than one hour. (Comparison is based upon alternate operation not being used at all.) The conjecture is confirmed very strongly; the look ahead heuristic is effective.

Conjecture 10: Re-do heuristic. This heuristic is effective by definition since the better of the two schedules is selected.

A test was run for static schedules to see if the second schedule on the average was better than the first. (Applies to rules 1-4 only.)

Results: The second schedule was somewhat better on the average but the improvement is not significant.

Conjecture 11: Alternate operation and look ahead heuristics used together. First a test was made to see that schedules laid out with both the alternate operation and look ahead heuristics are better than those with neither heuristic, for rules 1–6.

Results: For each of 25 static job files and each of the 16 dynamic job files the schedules improved on the average when the heuristics were used.

Then two more tests were conducted for the static schedules. First, the two heuristics used in ally vs. look ahead only: In only one of the 25 job files was the look ahead alone more effective on the average. Second, the two heuristics used in ally vs. alternate operation only: In 10 job files the results were identical; in 3, alternate operation was slightly better; in 12, the two heuristics performed better, on the average. The conjecture is sustained emphatically: The two heuristics used together are more effective than either one used alone, or neither of them used.

The status of our findings at this point is:

1) Non-random rules (such as rules 1–6) are significantly more effective than random rules.
2) There is little choice between rules based in some reasonable way upon job slack.
3) The shortest imminent operation rule is less effective than a job slack-based rule.
4) The alternate operation and look ahead heuristics are effective, both individually and collectively.

Thus several of the conjectures are concerned with what now appear to be unimportant issues, matters of differences between the rules; of importance is the behavior of the rules when strengthened by heuristics.

Based on the results so far, we venture this:

Conjecture 12: (a) Rules 1 through 6 are of approximately equal effectiveness when bolstered by the alternate operation and look ahead heuristics, and

(b) Each of rules 1 through 6 becomes more effective when used with the alternate operation and look ahead heuristics.

As a prelude, here is a listing of the six non-random rules, ranked 1 through 6 according to increasing tardiness hours, for each job file for the "blind" use of the rules, that is, without any heuristics. Figures show the sum of the ranks for all schedules.

Static Schedules ($N = 25$)		*Dynamic Schedules* ($N = 16$)	
Rule 1, job slack	69½	Rule 2	44½
Rule 2, job slack per operation	73	Rule 1	46
Rule 6, SIO-job slack ratio	78	Rule 6	51½
Rule 3, job slack ratio	92	Rule 4	57
Rule 4, modified job slack ratio	92½	Rule 3	60
Rule 5, shortest imminent operation	120	Rule 5	77

With the exception of rule 5, there is not a great difference in performance amongst the rules.

When both heuristics are employed, the rank sums are as follows:

Static Schedules ($N = 25$)		*Dynamic Schedules* ($N = 16$)	
Rule 2	65½	Rule 2	45½
1	78	3	47
5	83½	1	52
6	97	5	62
3	98½	4	64
4	102½	6	65½

The most notable change as compared with the blind use of the rules is that rule 5, shortest imminent operation, has returned to the fold as an effective rule. In addition rule 6, shortest imminent operation and job slack ratio, has moved downward in the rankings.

As a test of conjecture 12a, Wilcoxen tests were run between the "better" and "poorer" static rules (a posteriori), and only rule 2 could be found to be significantly better than any other rules, namely rules 6, 3, 4. For dynamic schedules, the difference between rules 2 and 6 was just significant at the .05 level. Thus conjecture 12a is not definitely refuted; more testing is necessary.

To test Conjecture 12b in the static case, two rules were selected: Rule 1, which appeared to be typical in the extent of improvement, and rule 4, which apparently made the least improvement. For both of these rules the improvement was highly significant. In the dynamic case rule 1 (typical) and rule 6 (least improvement) were tested; the improvement for rule 1 was highly significant, for rule 6, just significant at the .05 level. We conclude that Conjecture 12b is substantiated.

As a check on the statistical tests, consider the data tabulated below. Mean tardiness hours is a not unreasonable criterion, although schedules with more tardiness may carry too much weight in the results. Rules 1, 4, and 5 were selected as representative rules.

Heuristics Used	Mean Tardiness Hours					
	Static Schedules ($N = 25$)			Dynamic Schedules ($N = 16$)		
	Rule 1	Rule 4	Rule 5	Rule 1	Rule 4	Rule 5
None	18.7	26.5	36.7	72.2	84.4	177.2
Alternate operation and look ahead	13.0	19.4	14.4	64.2	60.2	70.1
Alternate operation, look ahead and re-do	10.8	16.0	14.4			

These figures indicate that perhaps there is a real difference in results attained with the use of different priority rules (even ignoring rule 5); however, the overall improvement—and rule 5 is the most striking example of this—is great enough that: First, a "poor" rule when augmented by the heuristics is better than a "good" rule without them, and second, the heuristics tend to reduce the differences in performance between rules.

Due to space limitations, computer output, job files, etc. are not shown here. Such information will be supplied upon request.

7. Conclusions

The immediate results for both static and dynamic scheduling problems are given in the preceding section. The twenty-five static problems involve 6 to 20 jobs, 1 to 16 operations per job, and 4 to 16 machines. The sixteen dynamic problems involve 20 to 60 jobs, 1 to 16 operations per job, and 4 to 16 machines. Two obvious points should be made:

—Not very many job files were used in the tests. A disproportionate number of them might possess some peculiarity which is significant in affecting the schedules.

—The same data were worked over a number of times in the attempt to detect statistical and practical significance, or lack of it. These items[17] should serve to temper with caution any conclusions which are drawn.

It does appear that the following conclusions are justified:

1) When the problem under study is the realistic one with the goal of meeting delivery commitments, the selection of a priority rule for discriminating between jobs competing for time on the machines is not as important as the selection of a set of heuristics which will bolster the rule.

[17] Another fact should be noted: From the data generated in the program, any extreme bottleneck problem was precluded. This was intentional; it is presumed (and hoped) that prior planning on the part of shop management has kept facility utilization somewhat balanced overall. Before jobs are admitted to the shop the total demand on each type of facility is known and the decision is made to work overtime, or subcontract, if necessary. If a severe bottleneck should exist, the strategy should be to get the overloaded machine in operation as soon as possible and keep it busy; none of the priority rules tested will necessarily accomplish this.

2) Since there is little difference in effectiveness of the priority rules after they are combined with two or more heuristics, a simple rule should be used.

Two simple rules are the job slack and shortest imminent operation rules. At the other extreme rather extensive calculations are entailed in the modified job slack ratio rule which takes machine loads into account. Except for an experimental research program, complex priority rules are inadvisable.

3) The heuristics which anticipate the future progress of a schedule, the alternate operation and look ahead heuristics (together with the insert routine), improve schedules significantly in both a statistical and practical sense, and the improvement in performance is ample reward for the incremental computing cost.

Although extensive tests are necessary to confirm this conclusion, it is probably true; the added computing time for both heuristics is typically only 10 to 20% of the total, for both static and dynamic schedules.

More testing is necessary to determine whether it is worthwhile to run off a schedule a second time after merely changing the target due dates for the late jobs. It may be that the computing time might better be spent on using another rule for the second schedule. Thus, although there is little difference in effectiveness between rules, for individual schedules marked differences often occur.

The implication of these results is that we should think more of matters other than straightforward priority rules for scheduling; for those interested only in operations as well as those performing research the way is clear for a number of investigations of the use of heuristic scheduling rules, which show great promise.

A final note: Although the heuristic program was designed for the due-date problem a few tests were made with the aim of minimizing the overall schedule time. This was accomplished by reading in the job file with all due dates identical and equal to the target schedule span. The procedure was repeated for several other targets. In the first set of tests, the data employed were those originally used by Giffler, VanNess, and Thompson (see [6]) for their Monte Carlo trials. Six job files were run off; in five of these, at least four of the six non-random rules, when used with the alternate operation and look shead heuristics, at least matched the best time found by the Monte Carlo program. For two of the job files the best Monte Carlo result was surpassed by the heuristic program; for one job file the Monte Carlo test time was one hour less than the minimum time from the heuristic program; for three others the minimum times were equal and it is known, or strongly suspected, that the time attained by both programs was the optimum.

One more test was made. In a paper by Heller [7] there appears a complete job file of 100 jobs, 10 operations per job, for a "flow shop" (each job goes to machines 1 through 10 in that order). Heller ran 3000 Monte Carlo trials for this job file and attained a minimum schedule time of approximately 575. The heuristic program was run with a target schedule time of 560. The minimum time was 559, for rule 5. The six rules respectively (with the heuristics) gave these results: 608, 608, 608, 599, 559, 561. Only a few of the Monte Carlo trials showed schedule times less than the greatest time, 608, resulting from the use of the heuristic program.

These trials indicate that the heuristic program may be very effective in handling the minimum make-span problem.[18]

Appendix A

Program Classification

This appears in the program as an array, "KC," with 25 variables. Not all variables are currently in use.

KC	*Job File:*	
1	Data origination	0 = Read in job file 1 = Generate job file
2	Machine groups	0 = No groups (each machine unique) i = Maximum number of machines in group; $1 < i < 10$
3	Job operations	0 = Each job has exactly KK operations 1 = Each job has 1 to KK operations
4	Job routing	0 = No more than one operation in a machine group 1 = May return to machine group 2 = All jobs have same routing, no more than one operation in a machine group
5	Bias in generated data	0 = No bias 1 = Bias toward certain machines (more heavily loaded)
6	Due dates	0 = As specified 1 = Due dates determined in program
	Scheduling Strategy:	
7	Job grouping	0 = Schedule all jobs together (not used) 1 = Schedule subset first (not used)
8	Replication of schedule	0 = Once only 1 = Schedule a second time with adjusted due dates.
9	Intraschedule change	(Not used)
	The Problem:	
10	Program employed	0 = Priority rule testing program 1 = Heuristic selection of rule to use (Not used) 2 = Minimum make span problem
11	Admission of jobs	0 = Static 1 = Dynamic
	Scheduling Strategy, cont'd.	
12	Priority rule (being used in current schedule)	i = Rule to use, $1 \leq i \leq 8$; applies only if KC(13) = 0. If KC(13) > 0, initial value of KC(12) is ignored, (let KC(12) = 0) During run, KC(12) = priority rule being used
13	Priority rule(s) to use	0 = Use the single rule indicated by KC(12) at start. i = Use each rule in turn up to and including i, $1 \leq i \leq 8$

[18] For a discussion of the minimum make-span problem, and an algorithm for its solution, see [5].

14	Time horizon	0 = Hour by hour only 1 = Hour by hour with "look ahead".
15	Control for alternate operations	0 = No alternate operations i = Use alternate operation routine on alternate schedule* 2 = Always use alternate operation routine (In dynamic schedule, (KC(15) = 0 or 2)
16	Alternate operations	0 = No alternate operations for this schedule 1 = Use alternate operation routine for this schedule (Irrelevant at start; let KC(16) = 0, initially)
17	Control for look ahead	0 = no effect 1 = Use look ahead routine on alternate schedules* (Not used)
18	Manipulation	(Not used)
19	Flexibility	(Not used)
20	Updating of priorities	Specified in program (Not used)
	Miscellaneous:	
21	Conditional print out ("low level")	0 = Print out everything 1 = Suppress conditional print out
22	Conditional print out ("high level")	0 = Print out everything 1 = Suppress conditional print out

* First the routine is not used in laying out the schedule, then the schedule is repeated (same priority rule) with the routine being used.

Appendix B

Formalization of the Priority Rules

We define terms as follows:

j = job number; $j = 1, 2, \cdots, J$
k = operation number (on job j), $k = 1, 2, \cdots, K$
n = number of next operation to be scheduled (on job j)
t = present time, hours
d_j = due date of job j, hours
h_{jk} = processing time (hours) for the k^{th} operation of the j^{th} job
h'_{jk} = "modified" processing time (hours) = h_{jk} plus the expected delay time
s_j = slack hours, job j
s'_j = "modified" slack hours, job j
j^* = job selected to be scheduled
$P^r_{j^*}$ = priority rating according to rule r for job j^*
ψ = the set of jobs waiting to be processed on the given machine at time t; ψ is a subset of J

The reason for calculating priority ratings which are described below is to determine what job is to be scheduled next on the given machine at the given time; the job with the minimum priority rating is the one selected. Thus the calculations need be carried out only for those jobs (the set ψ) which are waiting to be processed on the machine. The notation does not take machines into account, since the procedure is independent of the machine involved.

The first come-first serve and random priority rules are not included in this section.

Priority rule 1: Job slack

Job slack is the number of "free" hours available before the due date. If a job is necessarily going to be late, the slack is negative.

$$P^1_{j*} = \min_{j \in \psi} \{d_j - \textstyle\sum_{k=n}^{K} h_{jk} - t\}$$

Priority rule 2: Job slack per operation

Job slack per operation is the number of free hours available before the due date divided by the number of operations remaining. If a job is necessarily going to be late, the rating is the (negative) amount of slack.

$$P^2_{j*} = \min_{j \in \psi} \begin{cases} (d_j - \sum_{k=n}^{K} h_{jk} - t)/(K - n + 1) \\ \qquad\qquad = s_j/(K - n + 1) & \text{if} \quad s_j > 0 \\ s_j, \quad \text{otherwise} \end{cases}$$

Priority rule 3: Job slack ratio

Job slack ratio is the number of free hours available before the due date divided by the number of hours remaining until the due date. If a job is necessarily going to be late, the rating is the (negative) amount of slack.

$$P^3_{j*} = \min_{j \in \psi} \begin{cases} (d_j - \sum_{k=n}^{K} h_{jk} - t)/(d_j - t) = s_j/(d_j - t), & \text{if} \quad s_j > 0 \\ s_j, \quad \text{otherwise} \end{cases}$$

Priority rule 4: Modified job slack ratio

Modified job slack ratio is the number of free hours available before the due date, after operation times have been inflated to include expected delay times, divided by the number of hours remaining until the due date. If a job is necessarily going to be late, the rating is the (negative) amount of "modified" slack.

$$P^4_{j*} = \min_{j \in \psi} \begin{cases} (d_j - \sum_{k=n}^{K} h'_{jk} - t)/(d_j - t) = s'_j/(d_j - t), & \text{if} \quad s'_j > 0 \\ s'_j, \quad \text{otherwise} \end{cases}$$

Priority rule 5: Shortest imminent operation

Shortest imminent operation priority rating is the length in hours of the next operation of the job.

$$P^5_{j*} = \min_{j \in \psi} h_{jn}$$

Priority rule 6: Shortest imminent operation-job slack ratio

In determining shortest imminent operation-job slack ratio priority the job slack ratio is calculated for each job and a check is made to see whether any job is necessarily late (has negative slack); if so, the priority rating of each job equals job slack; if not, then the priority rating of each job whose slack ratio is no greater than twice that of the "tightest" job (job with minimum slack ratio) is equal to the length in hours of its next operation; the rating of any other job is irrelevant.

$$P^6_{j*} = \begin{cases} \min_{j\in\theta} h_{jn}, & \text{if } \min_{j\in\psi} s_j > 0 \\ \min_{j\in\psi} s_j, & \text{otherwise} \end{cases}$$

where θ is defined as the subset of jobs ψ for which $s_j/(d_j - t) \leqq 2 \min_{j\in\psi} \{s_j/(d_j - t)\}$.

References

1. Ackerman, S. S., "Even-Flow, A Scheduling Method for Reducing Lateness in Job Shops," *Management Technology*, Vol. 3, No. 1 (May 1963), 20–32.
2. Baker, C. T. and Dzielinski, B. P., "Simulation of a Simplified Job Shop," *Management Science*, Vol. 6 (1960), 311–323.
3. Conway, R. W., Johnson, B. M. and Maxwell, W. L., "A Queue Network Simulator for the IBM 650 and Burroughs 220," *Comm. Assn. for Computing Machinery*, Vol. 2, No. 12 (Dec. 1959).
4. —— and Maxwell, W. L., "Network Dispatching by the Shortest Operation Discipline," *Operations Research*, Vol. 10, No. 1 (Jan–Feb 1962), 51–73.
5. Gere, William S., Jr., "A Heuristic Approach to Job Shop Scheduling," unpublished Ph.D. thesis, Carnegie Institute of Technology, September 1962.
6. Giffler, B. and Thompson, G. L., "Algorithms for Solving Production Scheduling Problems," *Operations Research*, Vol. 8, No. 4 (July–Aug 1960), 487–503.
7. Heller, Jack, "Some Numerical Experiments for an *M* x *J* Flow Shop and Its Decision-Theoretical Aspects," *Operations Research*, Vol. 8, No. 2 (March–April 1960). 178–184.
8. Holt, C. C., "A Priority Rule for Minimizing the Cost of Queues in Machine Scheduling," Social Sciences Research Institute, University of Wisconsin, Dec. 1961 (ditto).
9. Karg, R. L. and Thompson, G. L., "A Heuristic Approach to Solving Travelling Salesman Problems," *Management Science*, Vol. 10, No. 2 (January 1964), 225–248.
10. Kuehn, A. A. and Hamburger, M. J., "A Heuristic Program for Locating Warehouses," *Management Science*, Vol. 9, No. 4 (July 1963), 643–666.
11. LeGrande, Earl, "The Development of a Factory Simulation System Using Actual Operating Data," *Management Technology*, Vol. 3, No. 1 (May 1963), 1–19.
12. McNaughton, R., "Scheduling with Deadlines and Loss Functions," *Management Science*, Vol. 6 (1959), 1–12.
13. Muth, John F., "The Effect of Uncertainty in Job Times on Optimal Schedules," O. N. R. Research Memorandum No. 88, Carnegie Institute of Technology, January 1962 (ditto).
14. —— and Thompson, G. L. (Eds.), *Industrial Scheduling*, Prentice-Hall, 1963.
15. Rowe, Alan J., "Toward a Theory of Scheduling," *Journal of Industrial Engineering*, Vol. 11, No. 2 (Mar.–Apr. 1960), 125–136.
16. Siegel, Sidney, *Nonparametric Statistics for the Behavioral Sciences*, McGraw-Hill, 1956.
17. Simon, H. A., *The New Science of Management Decision*, Hareper & Bros., 1960.
18. —— and Newell, A., "Heuristic Problem Solving: The Next Advance in Operations Research," *Operations Research*, Vol. 6, No. 1 (Jan.–Feb. 1958), 1–10.
19. —— and ——, "Computer Simulation of Human Thinking and Problem Solving," *Computers and Automation*, Vol. 10, No. 4 (April 1961), 18–26.
20. Sisson, R. L., "Sequencing Theory," Chapter 7 in Russell M. Ackoff (Ed.), *Progress in Operations Research*, Wiley, 1961.
21. Tonge, Fred, "The Use of Heuristic Programming in Management Science," *Management Science*, April 1961, 231–1237.