Workload Balancing and Inventory Minimization for Job Shops

Reprinted from
The Journal of INDUSTRIAL ENGINEERING
April, 1968

Workload Balancing and Inventory Minimization for Job Shops

George Schussel,[1] SENIOR MEMBER, AIIE
Manager, Information Systems Division, Brown Engineering, A Teledyne Company

The purpose of this article is to discuss a specific algorithm for forecasting and balancing the workload in a job shop. This algorithm provides a procedure for combining economic analysis and workload forecasts into an efficient economical schedule for a job shop.

The studies resulting in the preparation of this article were performed in an aerospace company possessing a rather large and complicated job shop. The complexity of the jobs passing through the job shop was such that in some cases six to nine months of flow time was required to process a single job in its entirety. In almost all cases, the planning and release of orders occurred months ahead of the actual time that the finished product was needed, because no method of forecasting and controlling the workload in the job shop existed, and, in order to provide efficient utilization of labor and machines, it was necessary to have a large inventory of work waiting behind every machine on the floor. This unnecessarily large inventory enabled most machines to be used continually and most jobs to be turned out on a timely basis. This situation of excess inventory and its associated carrying cost is typical in the aerospace industry and may be to a limited extent characteristic of all job shops.

[1] Formerly Executive Assistant to the Corporate Manager of Management Information and Data Processing, Northrop Corporation.

Management determined the release dates of orders so that there was almost always plenty of work for every machine to perform and almost all work was performed on time. Whenever a machine would encounter an idle period, its operator would simply reach forward into time and complete work ahead of schedule. As a result of this policy, it was felt that, with a better method of forecasting the individual workload on each type of machine and with a method of leveling out the forecasted peaks and valleys of utilization on each machine, it would be possible to maintain efficient and timely production in the shop with a great reduction in inventory levels. By controlling the release dates and cutting back on the early order release times, management can reduce the cost of maintaining inventory in a job shop. If the average time for an order to pass through the job shop is reduced by a certain percentage, it has the same effect on the costs of maintaining inventory as reducing the average inventory level by the same percentage.

Accordingly, this article presents a method of manipulating and handling information relating to the forecasts of workloads in a job shop. It does not discuss priority rules and other subproblems of running a job shop. Many authors have written ample amounts on these various problems. This article discusses the general problem of scheduling and workload balancing in a large job shop and offers an algorithm that performs these functions and is in such a form that a computer program can be written fairly simply from the steps mentioned in the algorithm. The objective is to determine latest release dates that are consistent with schedule requirements. A survey of the literature shows that the method presented here uses less restrictive assumptions and handles more relevant variables on a heuristic basis than any other presentation the author has seen.

The basic approach discussed here is to consider the representation of information as a two-dimensional matrix. Along the vertical axis of this matrix we find the various machines or subassembly points of the job shop listed, while along the horizontal axis we have equal increments of a time unit marked off. For the purposes of this article, we will assume this time unit to be one day although it could also be one week in a very large job shop where the primary purpose of this algorithm would be to determine the release dates of each order.

The vertical coordinate on the matrix represents all of the shop's machines or load points and the horizontal coordinate covers the total period of time scheduled. Each cell of the matrix can be viewed as a pigeonhole in which we store relevant information concerning work to be done on a machine at a cer-

tain time. This information consists of items such as the job numbers that have been assigned to each cell and the hours of work each is expected to take. Once we have this matrix, the basic procedure is to load jobs into it as if we had unlimited capacity in each cell of the matrix. After this initial loading is completed, another pass through the matrix is performed, this time with the objective of smoothing out the uneven workload which resulted from the initial loading.

OBJECTIVES

Several major objectives can be met by the implementation of the scheduling method outlined in this article.

1. The use of the routine outlined below in conjunction with powerful modern computers will permit long-range workload scheduling that is typically not now possible for most firms. All of the advantages of being able to forecast or anticipate future workload will accrue to the user of such a routine.

2. Through the use of cost calculations and heuristic logic, it is possible to attempt to satisfy several different objectives. One of the main reasons one does not find the job shop problem "solved" in the literature is that before a problem may be explicitly solved it normally has to be formulated in terms of an objective function which can be maximized or minimized. The trouble with the problem of scheduling a job shop is that there are many different objectives which can and must be met in order to have an efficient flow of work through the job shop and operate at minimal cost. For example, one may wish to minimize any one or some combination of the following variables: average job lateness variance of job lateness, average work-in-process inventory, total setup time, total labor cost, or machine idle time.

By using cost calculations to help level (or balance) out the workload, the routine achieves the objective of making efficient use of labor and machine productive capabilities. In addition to efficient usage of shop capabilities, however, via the process of compressing work schedules and scheduling backwards, the routine attempts to satisfy a primary objective of minimizing work-in-process inventory carrying costs. The true realization of lower inventory carrying costs, however, only comes about through the shorter flow times that are possible when management uses the improved schedule information to reduce flow times and eliminate unnecessarily large safety factors.

3. Another important use of the workload balancing scheduler is to generate vastly improved management information over what is typically available to job shop management. Through the use of the evaluative power of a computer, routine cost calculations, that would be impossible for clerks to perform, can become a regular part of production scheduling. Freed from the necessity of performing these calculations, management can focus attention on important problems that are beyond the scope of the computer program to handle. The system should, however, pick out these problems from the large amount of minute detail required for scheduling and present them in a form that is amenable to management analysis. Also, because the workload balancing scheduler performs calculations for a time period in the future, the potential problem areas can be highlighted early enough for management to take action to correct them.

IMPORTANT CONCEPTS

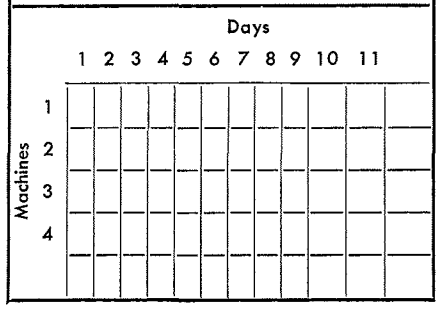
Three concepts are of basic importance to the algorithm presented below, and these concepts are discussed here in order to give the reader more insight into the way they are used in the scheduling routine.

Matrix Representation of Information

The first of these concepts is the idea of the familiar Gantt chart or what is here referred to as the two-dimensional matrix (Matrix 1).

With days listed across the top and machines listed along the side, a row out of the matrix may be interpreted as the daily schedule for one machine over a period of days while a column

Matrix 1. Information array



from the matrix can be considered as one day's schedule across all of the machine or subassembly loading points in the shop. As each job is scheduled into a cell or combination of cells, certain relevant information about that job, such as its number, any predecessor operation and relevant cost information, is stored in the cell. If the total number of days we are scheduling is D, the total number of machines or load points is M, and the total amount of information stored in each cell is N, then we have a $D \times M \times N$ size array of information. This matrix represents a dispersion of the necessary information for accomplishing scheduling and loading. This matrix is used as part of the algorithm for laying out the production schedule for a generalized job shop. This requires the capability of maintaining the strict sequencing of a part from one machine to the next. However, the typical assumption that any product may only be made by one strict sequence of machines is not entirely required. By evaluating which of two alternative identical machines has the lighter load, the algorithm can effectively evaluate different production machine sequences. The general problem of evaluating different productive sequences over different non-identical paths for each product in a job shop, however, presents a combinatorial problem of such magnitude that no formal solutions exist.

The use of this matrix also enables several economic considerations to be analyzed. Part-to-part comparisons can be made when a machine is overloaded in order to choose the best part to reschedule. Since we are working backwards from fixed delivery dates, the method presented in this article as-

sumes that any rescheduling of a part can only be done earlier in time, since most managements would not accept a scheduling method which had a designed-in capability for late deliveries. Therefore, every time that a job is rescheduled from the unlimited capacity matrix, a cost due to overtime pay or increased inventory from a longer flow time is incurred. As a secondary objective, the scheduling algorithm can evaluate the costs due to schedule changes and print out a list of those jobs which incur the highest percentage increase in costs because of insufficient in-house capacity. Via this procedure, the computer can print out a list of problem exceptions that it was not programmed to handle automatically. For example, if the accomplishment of a certain job required a large amount of overtime, this fact would be brought to management's attention via a print-out list. This job could then be analyzed by planners to see if it could be subcontracted.

Priority Dispatching Rule

A second important concept in the scheduling algorithm is the use of a priority dispatching rule to determine which jobs should be rescheduled when an overloaded condition exists on any work facility.

Priority dispatching rules are typically used in a different context. Much work has been done toward determining which heuristic rule provides the best priority dispatching ranking for job shops. Typical of research results are works by Conway and Gere (4), (5), and (7). In the context discussed by these authors, a priority dispatching rule is used to determine the order in which jobs waiting in line at a facility should be processed.

By definition in the scheduling algorithm, no job is late because jobs are scheduled backwards from on-time dates allowing sufficient setback times for work flow, travel time, etc. Nevertheless, a priority ranking rule can be applied to the jobs in a given work cell and even though no job is late, a ranking will result. The highest-ranked jobs will be those that are the nearest to being late. Accordingly, the scheduling algorithm uses a priority ranking rule to decide which of those jobs in an overloaded cell should be rescheduled earlier in time. The job that has the highest priority is closest to being late and therefore is the one which is chosen to be released earlier in time, hopefully reducing the chances that it will be late. The use of such a priority rule for choosing which jobs are to be rescheduled helps the scheduling method to satisfy another objective, that of minimizing the late time of the jobs processed through the job shop.

The priority dispatching rule used in Step 3 of the scheduling algorithm is based upon the work reported by Conway in (5). This work evaluated most common dispatching rules and suggested that a combination of the minimum slack time per operation rule and the minimum imminent processing time rule would be the best. His analysis indicated that an equally weighted linear combination of the two rules provided a new rule which is generally optimum:

$$\text{Priority} = P_t + ST/O.$$

P_t is the processing time on the imminent operation and ST/O is the slack time per operation remaining. This slack time is equal to the total schedule time until due date minus the total remaining scheduled work time for the release. The highest priority is achieved by that release with the smallest numerical ranking via the priority rule. It is important to notice that exactly which priority rule or what criteria are used to determine which job is relocated is not important to the central theme of this article. As long as we have some manner of picking among the jobs we can proceed to applying the algorithm.

Logical Time Period

Another important concept in the scheduling procedure is that of the LTP or logical time period. The need for such an entity arises because a realistic simulation has to take into account job continuity from one day until the next. A daily period of time is an arbitrary measure and there may often be orders that require several days of work on any one machine or that are only partially completed in one day. An LTP is a series (one or greater) of adjacent whole days which are so defined that any job operation in the LTP both starts and finishes in the LTP. Of course, an individual day may be an LTP and is the smallest such unit. The LTP provides the basic entity to which the scheduling algorithm allocates jobs. The advantage of using the LTP concept is that it permits simplified handling of the job continuity problem.

Since the workload balancing scheduler is a long-range forecast device, whose primary purpose is to determine release dates for orders, it has no need to schedule to very small periods of time, such as an hour. By constantly scheduling with safety pad times, a small amount of juggling of schedules is possible by floor supervision. This fact makes it possible for the algorithm to ignore the ordering of jobs on an intra-LTP basis in small LTP's. When we are rescheduling a very large job that runs for several days on a machine, we look for or create a new LTP which is several days long and can accommodate this job in addition to the other assorted jobs already scheduled in the period. This feature is necessary if large holes of available time are not to be left in the schedule because of rescheduling maneuvers.

When the scheduling algorithm is loading jobs that require a small percentage of the daily available time on any machine, it ignores the problem of continuity of jobs within a day. This will be taken care of by the detailed scheduling done by shop foremen. However, when scheduling a job that takes greater than 25 percent of the daily capacity on any machine, the algorithm searches for two or more adjacent days or LTP's having a sum of available times to work the job. The length of the period of time searched depends upon the percentage of daily capacity required by the job. For example, if machine capacity of 80 percent of one day's time is required on an order, then the algorithm might search for two adjacent days which have this time available. These days

could be combined into an LTP and the job loaded at the end of the LTP. The smaller jobs from this last day would be rescheduled into the first day (on an LTP of less than seven days length, this step would not be performed until the very last step in the algorithm). By the use of the LTP, the scheduling algorithm allows for the possibility of small changes without having to reflect the small changes through the entire system. Without the use of a concept analogous to the LTP, a computer program that performed workload scheduling and forecasting would be much more complex than the one presented in this article. The concept of the LTP will become clear as the reader proceeds forward through the scheduling algorithm.

DEFINITIONS

The following definitions are presented in logical groupings. The definitions should be read carefully before proceeding to the algorithm.

Index Numbers

d = day number. This variable performs an indexing function.

D_i = list of completion due dates for each release of a product. This variable is part of the input information.

i = order number. An order is a group of identical parts or products that are made at the same time and follow the same routing through the job shop. Index variable having a one-to-one correspondence with order numbers.

j = machine number. Index variable having a one-to-one correspondence with an individual machine or subassembly operation.[2]

J_i = a series of K_i dimensional vectors giving the logical machine sequence for each release. This variable is part of the input information. Each element of J_i is j and is assumed unique. The logic of the scheduling algorithm, however, can be expanded to include evaluations of alternative j's for the same k.

k = the index of operations running from 1 through K. This variable performs an indexing function.

K_i = number of operations required for ith release. Calculated variable derived from J_i.

task (i, k) = a task. Refers to the kth operation on order i. An operation is a function performed by a machine or subassembly point to an order.

Schedule and Capacity Records

C_{jd} = regular time capacity matrix in hours per day. (The amount of time available any particular day on a machine before it is loaded.) Special factors such as vacations and reduced efficiency are included. The normal conditions capacity is C_j. This variable is part of the input information.

O_{jd} = overtime capacity matrix. This is analogous to C_{jd} except it pertains to the overtime capability. The normal overtime capacity is O_j. This variable is part of the input information.

L_{jd} = unlimited capacity initial loading matrix. Variable used in calculation. The final value for L_{jd} is the forecasted, balanced workload. Output and input variable.

G_{jd} = committed overtime capacity matrix. The final value for this is the overtime schedule. This variable is part of the output information.

H_{ik} = setup and labor production hours of a task (i, k). This variable is part of the input information.

Rescheduling Operations

LTP = logical time period. The LTP consists of a series (one or greater) of adjacent whole days which are so defined such that any task (i, k) in this LTP both starts and finishes in the LTP. An individual day may be an LTP and is the smallest such unit. The LTP provides the basic entity to which the scheduling algorithm allocates jobs. Until the last step of the scheduling algorithm LTP's are not subdivided, although they may be combined so that two or more LTP's form one new larger one.[3]

n = the length of an LTP. It is the number of days in the LTP.

s = the minimum length (in days) of the requisite adjacent LTP's used in the search process of relocating a task (i, k).

z_j = percentage/100 of daily capacity on an operation. If rescheduling the top priority job causes a greater idle factor than z, the other jobs (or combinations of jobs) should be analyzed for rescheduling. This is an input variable which is best set by experimentation on the system.

Cost Evaluations

a = number of days of early

[2] The terminology used in this article is from Gere (7) and others: "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." A statement such as the following can be made: the task (i, k) which consists of the kth operation on the ith release or order must be performed on the jth machine.

[3] Because an LTP can cover several days, there may be several values for the index "d" that correspond to one LTP.

release required to schedule an order on any operation. (Incremental days for this operation, not cumulative.) This variable is used in calculating A.

M_{ik} = material costs of any release at any operation. (Incremental, not cumulative.) This variable is part of the input information.

r_j = hourly rate applying to the jth labor operation (including variable overhead). This variable is part of the input information.

$$V_{ik} = \sum_{k=1}^{k} [r_j H_{ik} + M_{ik}],$$

j corresponding to each k. Value in dollars of each release up through the kth operation. When $k = K_i$, V is the final value. Variable used in calculation.

A = loss associated with rescheduling an operation earlier. (Variable used in calculation—does not need to be stored as a matrix.) This is the loss associated with having only a limited production capability. The definition of A and the manner in which it is used is brought out in the following paragraphs.

Derivation of A

Where the available capacity is the unscheduled production or setup time on an operation any day, the available regular hours are $C_{jd} - L_{jd}$. The available overtime hours any day are $O_{jd} - G_{jd}$.

When $L_{jd} > C_{jd}$, the algorithm searches backwards (starting with the current day) until a day is found where

$$C_{jd} - L_{jd} + O_{jd} - C_{jd} \geq H_{ik}.$$

A day that satisfies this criterion possesses adequate unused capacity to process the job operation.

Or, if $H_{ik} > \frac{1}{4} C_j$, search for:

$$\sum_{d=w}^{y} [C_{jd} - L_{jd} + O_{jd} - G_{jd}] \geq H_{ik'}$$

an available time slot (regular and overtime) in several adjacent days (w to y). (The $\frac{1}{4}$ figure is arbitrary).

If d_{ik} is the day that the scheduling algorithm is working on, then "w" and "y" vary backwards from "$d_{ik} - s + 1$" and "d_{ik}," respectively, always maintaining a numerical difference of "$s - 1$." (To understand how this is modified to account for the LTP concept, see Step 7 of the algorithm.)

If the job is scheduled into this slot, define the hours worked of task (i, k) during regular time as

$$h_1 = \min \left[H_{ik}, \sum_{d=w}^{y} (C_{jd} - L_{jd}) \right]$$

and the overtime hours as $h_2 = H_{ik} - h_1$.

This definition causes all of the regular time hours to be used up before any overtime is scheduled.

With these definitions and A defined as the loss associated with any change in schedule caused by $L_{jd} > C_{jd}$, we get

$$A = .001a[V_{ik-1} + M_{ik} + r_j(h_1 + 1.5h_2)] + 1.5 r_j h_2.$$

The .001 is the percentage loss per day of early release, assuming a 25 percent per year carrying cost and 250 working days in a year. The term within the brackets is the sum of the expenditures on labor and material through the current operation. Since all of these expenditures are moved forward in time and "a" is the number of days of early release, multiplying by $.001a$ gives an expression for the loss associated with carrying the inventory additional time. All calculations involving A or any other expression that requires time should use job start date as the relevant time criteria. The final term in the expression for A is the loss associated with overtime payments. The total overtime schedule cost is assumed to be variable according to the number of hours scheduled. For incremental analysis, the regular time payroll is considered fixed.

The expression for A does not take into account cost changes which may result from the required rescheduling of predecessor operations of task (i, k). Costs due to schedule changes in the predecessor operations, such as additional or less overtime or early releases, are eliminated by assuming that over a large number of rescheduled items the costs and savings due to these changes balance to zero.

The assumption implies that if an operation is rescheduled earlier, all of the predecessor operations for this order must be moved forward on the average the same number of days early. In some cases, this may understate the amount of earliness required in earlier releases because jobs may now be scheduled into full capacity periods, where before the rescheduling they were not; or, conversely, it could overstate the amount of early releases required because capacity might now exist at minimum setbacks where it did not under the earlier schedule.

The primary benefit of this assumption is that it obviates the need to examine the cost ramifications of a schedule change through all predecessor jobs. In any case, it should be clear that an early release at operation $k - y$ for release i must be at least as great

Table 1. Definition listed alphabetically

- a = number of days of early release required
- A = loss associated with rescheduling
- C_{jd} = regular capacity matrix
- d = day number
- D_i = list of completion due dates
- G_{jd} = committed overtime capacity matrix
- H_{ik} = setup and labor production hours of task (i, k)
- h_1 = regular hours for any slot
- h_2 = overtime hours for any slot
- i = release or order number
- j = operation number
- J_i = logical operation sequence for each release
- k = the index of operations
- K_i = number of operations required for ith release
- L_{jd} = unlimited capacity initial loading matrix (becomes regular committed capacity matrix)
- LTP = logical time period
- M_{ik} = material costs
- n = the length of an LTP
- O_{jd} = overtime capacity matrix
- r_j = hourly rate
- s = the minimum search length of adjacent LTP's
- task (i, k) = refers to the kth operation on release i
- V_{ij} = value of each release up through the kth operation
- z_j = percentage/100 of daily capacity

as the early release at operation k. This is because there always must be a minimum setback between operations. The preceding assumption does not contradict this statement.

SCHEDULING ALGORITHM

Initial Load and Search

1. From D_i and the associated setbacks, and the current in-house work, load L_{jd} with all i for all d we are considering. Assign permanent large number priorities to current work in process. At the end of this step all of the day units in L_{jd} are LTP's with $n=1$.

2. Starting from the largest d and going across all j and then d, test until the first

$$\sum_{d=w}^{y} (L_{jd} + G_{jd}) > \sum_{d=w}^{y} C_{jd}$$

is located.

Pick Job[4]

3. For all orders in this LTP, calculate the P_t+ST/O priority ranking.

4. For the top priority (smallest number) order, test to see if rescheduling causes L_{jd} to have an idle capacity greater than z_j.

5. If the idle capacity is greater than z_j, search down the priority list until an order is found, the rescheduling of which would cause an idle capacity of less than z_j. If none available, pick the order which causes the smallest idle capacity.

Find Cheapest Reschedule

6. Determine the search length, s, from H_{ik} and the equation below, $(X=H_{ik})$ rounding off L to the nearest integer, s.

$$L = .65935 + 2.5500269X - .26599786X^2 + .029138699X^3 - .0014992671X^4 + .000035603459X^5 - .00000031582326X^6.$$

This equation was derived by fitting a polynomial to an arbitrary set of reasonable search length values. Some typical values are listed below.[5]

H_{ik}	L	s
1	2.97	3
2	4.90	5
4	8.12	8
12	20.13	20

(If $X \leq \frac{1}{4}$ set $S=1$.)
(If $X>37$ set $S=1.3X$, rounded)

7. Search backwards (starting with current LTP) over adjacent (one or more) LTP's whose sum of "n" (days) is minimally greater than or equal to "s" for enough available free time to schedule task (i, k). Where "w" is defined as the start date of an LTP and "y" is the end date of an LTP, this search condition is defined as satisyfing

$$\sum_{d=w}^{y} (C_{jd} - L_{jd} + O_{jd} - G_{jd}) \geq H_{ik},$$

an available time slot in one or more LTP's made up of the adjacent working days "w" to "y". The "minimally greater than" condition can be achieved by adding, one at a time, LTP's from the front (earlier time) and removing LTP's from the back (later time) as much as is possible without violating the above constraint. If no LTP with adequate space is found, return to Step 4 and pick another job with less hours than this task.

8. Calculate A for this task (i, k) and the LTP that would be created by combining the adjacent LTP's with adequate space into a new single LTP.

9. Continue backward searching and evaluating A's until an LTP (or combination of) is found where $h_i \geq H_{ik}$ or until the first day of the schedule is reached. The search process for available time capacity slots stops at this point.

10. Of the feasible slots just evaluated, pick the smallest A and associated LTP's and schedule the order into that slot by creating a new LTP which is equal to the sum of these other LTP's. Place all of the jobs from the wiped out LTP's into this new one in addition to placing the rescheduled job in the newly created LTP. If no slot with adequate capacity is found, go to a special routine which could handle this special case. (The sum of LTP's may, of course, refer to only one.)

11. If the length, n, of the new LTP is greater than six days, (an arbitrary choice) rank all of the jobs in it by the priority criterion. Consider the entire LTP as a continuous time period from w to y; starting from y and the jobs with the largest numerical priority, place them in the LTP from the back (y) toward the front (w). For each task (i, k) in the LTP, retain the start date before the creation of the new LTP and the start date after this continuous scheduling process internal to the new LTP. For any task (i,k) whose start date has changed, reload its predecessors the same number of days early (a) in L_{jd}. This criterion of change also applies to the new task (i, k) that was placed in this LTP. The process of continuously loading from the back will shift all of the idle time in this LTP to the front. If this idle time is greater than or equal to one day, cut off one day at a time from this LTP and form new LTP's of one day's length until the remaining idle time is reduced below one day.

12. Return to Step 4 and continue at the same pigeonhole until

$$L_{jd} + G_{jd} \leq C_{jd} + O_{jd}.$$

13. When the test in Step 12 is passed, return to Step 2 and continue to iterate through the algorithm. Step 2 this time starts from the last pigeonhole (LTP) that the algorithm was working on.

14. The procedure is complete when all k_i operations for each i have been processed. The schedule of work is L_{jd}, G_{jd} and associated information. The resulting schedule may violate some start day constraints because it requires the start date on a release to be earlier than is feasible (for example,

[4] There are several reasonable procedures by which a job can be picked for rescheduling. The method presented here has the objective of minimizing average job lateness. Other procedures for picking a job are available and might be more appropriate for certain situations. One such method might involve criteria that seek to determine schedules that are as level as possible.

[5] These values are strictly informal in the sense that they attempt to be reasonable search lengths for jobs of different lengths. If the search length is too short, our schedule may leave large amounts of unused time and if s is too long, we may have to disrupt our current schedule too much just to fit in one more job.

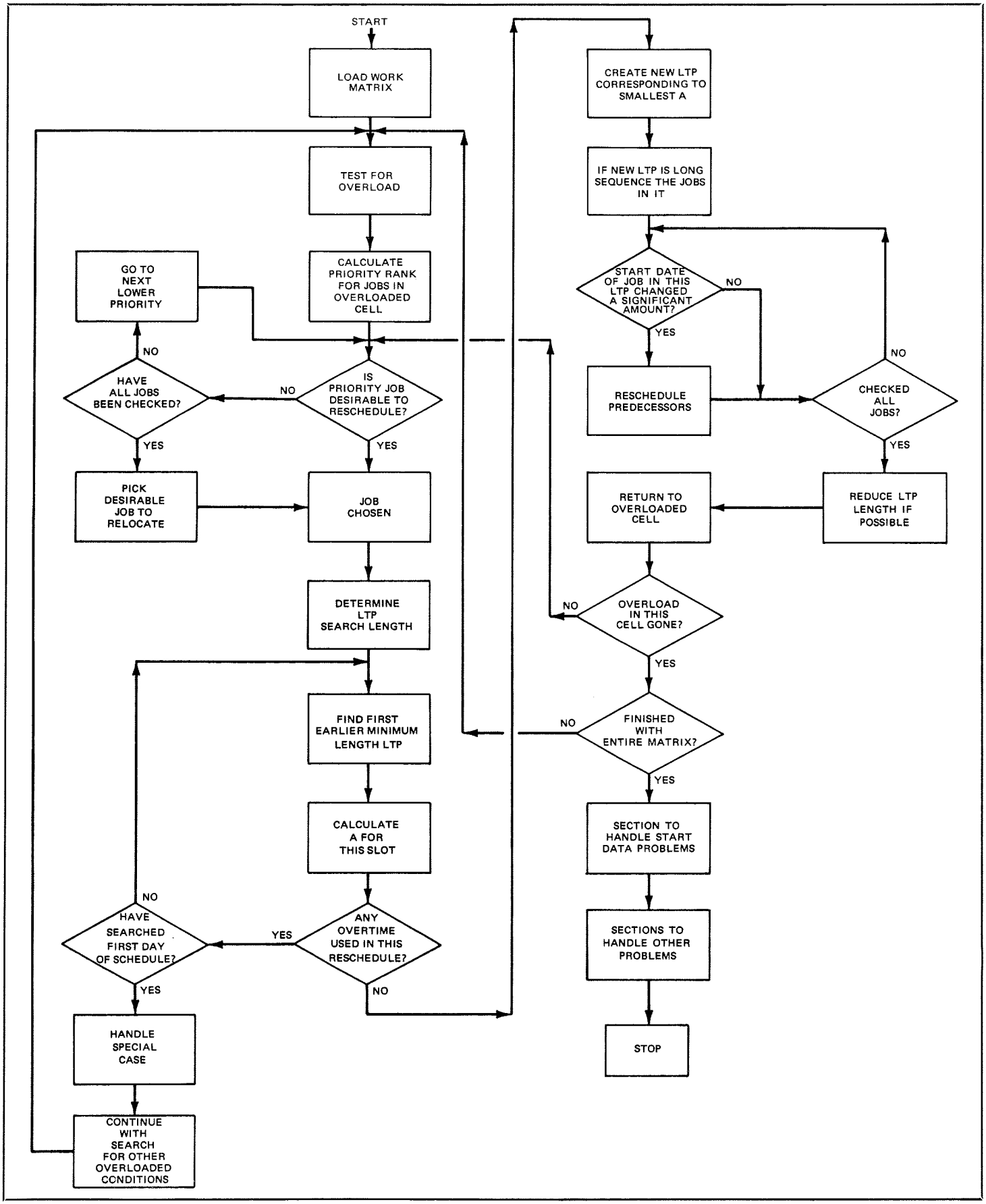


Figure 1. Scheduling algorithm

before the necessary raw material is scheduled for delivery). These violations could be handled by the following section.

Start Data Violation

15. Test for violation of start date constraints.

16. For every violation check to see if the scheduled required flow time $>$available time. If so, print out an infeasibility notice and proceed.

17. If a feasible schedule is possible (flow time $\leq$available time) print out the current schedule so a record is maintained.

18. For every violation, starting from start date as fixed, reschedule the violating orders forward into L_{jd} and fix (cannot be moved—assign permanent large value priority number) on all operations.

19. Go to Step 2 of algorithm and proceed backwards through the schedule again.

20. Stop after a final schedule that does not violate start date constraints is determined or after a given number of cycles.

21. When the scheduling process is finished, a schedule will be available in terms of LTP's instead of days. If a daily schedule is desired, it may be obtained by placing the job times within the LTP end to end as in a Gantt chart. The daily schedule can then be read off the chart.[6]

Adjustment for Limited Rapid-Access Computer Memory

Third generation computer equipment has brought a vast increase in the amount of memory storage available to the computer programmer. Large-core storage, drum storages and disk storages are typical of the types of devices that are used to store a large amount of information. However, even with these added capabilities in terms of memory storage, many significant job shop operations would be too large to have all resident data residing in high-speed memory. The algorithm did not pay any attention to the problems of maneuvering data among the various memory facilities of a computer and the following section is devoted to presenting a method by which the algorithm could be implemented on a computer having a limited amount of high-speed core memory plus an additional amount of slower memory such as disk or tape which is capable of storing all of the required information.

1. The first step in the process is to load the unlimited capacity matrix as described in the scheduling algorithm. The same process as that described below could be used to perform this task.

2. Once the entire unlimited capacity matrix has been loaded and exists sequentially arranged in disk or tape storage, we proceed to the load leveling process.

3. The first step is to bring in the entire first row; in other words, to bring in all the cells corresponding to one machine for all days. The last day in the row is checked to see if it is overloaded. If it is not, we simply put this information back on the disk or tape and bring in the second machine's complete row. As we proceed down the list in this fashion, we will run across a day-machine combination which is overloaded.

4. We now proceed to level the load on this day by shifting some of the jobs up in time according to the criteria presented in the scheduling algorithm. Since all the days for this one machine are in core memory, we can do this rapidly. However, the process cannot be completed because those jobs which are rescheduled earlier and have predecessors will typically have their predecessors on machines which are still on disk or tape storage. Therefore, when such a predecessor requires change, it is put on a list of changes to be made when the machine to which the change pertains arrives in core. The idea of using this list as intermediate storage is the key element to the present procedure. Only the job identification and the number of days earlier it is rescheduled must be stored. As previously mentioned, this rescheduling movement of a predecessor may create a newly overloaded day. Since any such overloading is earlier in time, it will be leveled out later in the process.

5. When the last day in the machine row being analyzed is no longer overloaded, it is put into disk storage and replaced in core by the next machine row. The list of stored items is then examined to see what changes apply to the new machine in core. These changes are made and the last day of this machine row is examined to see if it is overloaded (as we did in Step 3). This process is continued through the entire matrix proceeding across all machines at any one day, then moving one day earlier and going down all machines. Each successive time a row is brought into core, the last day from the previous time is not brought in because it is already leveled. It is obvious that if there is a total of M machines the longest an item can stay on the list is M-1 cycles. The average item on the list will remain on the list $M/2$ times before it is unloaded into its correct machine row. If the machines are grouped according to some logical sequence and numbered, we can reduce this figure considerably below the expected $M/2$ figure for random ordering.

As each cell is left, it is not overloaded and can not become overloaded because any subsequent changes in predecessors and basic scheduling involve only days that are earlier in time that will be reached as the algorithm moves toward earlier days.

For purposes of simplicity, we have treated each day as if it were scheduled independently, ignoring the LTP concept discussed earlier. It turns out, however, that this presents no problem if the LTP is brought into core as a whole until every day that it covers is later than the numerical day on which the algorithm is working. If the LTP is n days long, it will be processed n times as this method passes each machine day by day earlier. The reshuffling of the jobs in the LTP as suggested in Step 11 of the scheduling algorithm should not be done until the last time through the LTP. If the above method were implemented on a third generation high-speed processor, the procedures would typically be limited by the access time to the slower disk (or other)

[6] As a result of the above process, all excess work will be shifted to the front of the matrix.

storage. If the matrix is D days long by M machines deep, and we have enough high-speed memory capacity to access q rows at a time instead of one row at a time as suggested above, the total number of accesses to the slower medium would be $(D \times M)/q$ times for each pass through the matrix. The basic scheduling method, without adjustments, involves two passes through the matrix. Therefore, if each access to disk and transfer required p seconds, the total time for the routine would be

$$\text{Computer Run Time} = \frac{2p(D \times M)}{q} .$$

If, as assumed, the modified scheduling method just presented in combination with a computer is search and access time bound, it is clear that, if transfer rates from peripheral storage remain very high as high-speed core (or other) memory is doubled, the time to perform the scheduling algorithm will be halved because twice as many rows can be entered and accessed at any one time.

CONCLUSIONS

In the aerospace industry, management typically makes up for a lack of control and forecasting ability by releasing orders very early and having extra large amounts of inventory present in the job shop. As the management of such a job shop acquired experience in the use of a forecasting and workload balancing routine, the level of uncertainty about future schedules in the job shop would diminish and the amount of pad time or inventory back-up needed at each operation to compensate for this uncertainty should become much less. Over the long run, this experience with the scheduling method would instill confidence and provide savings through reduced production and inventory costs for the typical job shop. The lower inventory costs would be achieved by the reduced flow times and faster inventory turnover, while more efficient production would occur through improved workflows.

Low work-in-process carrying costs can be achieved only by shortening flow times and performing production operations as late as possible without impairing delivery schedules. In order to accomplish this objective, the scheduling algorithm starts from a given due date and schedules backwards, earlier in time, with a setback procedure for every part. Not only would the length of this setback become shorter as management acquired experience with the use of the scheduling routine, but because all of the work in the shop is done as late as possible, the excess capacity on each machine will tend to be moved towards the start of the period under consideration. One result might be that the first implemented schedules, determined by this algorithm would cause certain machines to have idle periods for the first days in the schedule. This would be true even though safety factors were built in to allow for unforeseen slippages. Management would then have a decision as to whether it is more desirable to carry excess inventory or to have idle labor. There is no way out of this conflict; the very process of efficiently scheduled production insures that each time a new schedule is determined the excess capacity built up from previous schedules is brought forward. However, with the aid of long-range workload balancing and scheduling, it may be possible for management to analyze future schedules for the possibility of obtaining additional work to fill periods of excess capacity.

Two final limitations of the scheduling method presented here should be mentioned. The first is that the actual schedules produced by the workload balancing algorithm will not be valid for the latest days in the schedule. These days are underloaded because they include only those projects that are just finishing. The second limitation has already been mentioned and concerns the point that the suggested algorithm ignores the existence of the possibility of manufacturing a part or product by more than one machine sequence. It assumes that the vector, J, is unique for each order release. However, if the difference between two alternative sequences that can be used to manufacture the part can be represented simply by one-for-one substitutions (for example, this would hold true for different yet identical machines), this assumption can be modified. A section in the algorithm could evaluate which of several identical machines is least loaded and would be the best to process an operation on a part.

REFERENCES

(1) Bulkin, Colley, Steinhoff, "Load Forecasting, Priority Sequencing, and Simulation in a Job Shop Control System," *Management Science*, October, 1966.

(2) Calica, A. B., "Production Order Sequencing," *IBM Systems Journal*, Volume 4, No. 3, 1965.

(3) Cobham, A., "Priority Assignment in Waiting Line Problems," *Operations Research*, Volume 2, No. 1, February, 1954.

(4) Conway, Richard W., "Priority Dispatching and Job Lateness in a Job Shop," *Journal of Industrial Engineering*, Volume XVI, No. 4, July–August, 1965.

(5) Conway, Richard W., "Priority Dispatching and Work-in-Process Inventory in a Job Shop," *Journal of Industrial Engineering*, Volume XVI, No. 2, March–April, 1965, pp. 123–130.

(6) Conway, R. W., Maxwell, W. L., and Miller, L. W., *Theory of Scheduling*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1967.

(7) Gere, William S., Jr., "Heuristics in Job-Shop Scheduling," *Management Science*, November, 1966.

(8) Huish, D. R., "Rescheduling Flattens Zig-Zag Production Curve," *American Machinist*, July 7, 1952.

(9) Johnson, S. M., "Optimal Two and Three Stage Production Scheduling," *Naval Research Logistics Quarterly*, Volume I, No. 1.

(10) Kilbridge, M. and Webster, Leon, "A Method of Solving a Class of Scheduling Problems," *Operations Research*, Fall, 1960, 8:Supplement 2:B-131 (Abstract).

(11) Magee, J. F., *Production Planning and Inventory Control*, McGraw-Hill Book Company, New York, 1958.

(12) Manne, Alan S., "On the Job Shop Scheduling Problems," *Operations Research*, March–April, 1960.

(13) Muth, J. F., and Thompson, G. L., Editors, *Industrial Scheduling*, Prentice-Hall, Englewood Cliffs, New Jersey, 1963.

(14) Page, E. S., "An Approach to the Scheduling of Jobs on Machines," *Journal of the Royal Statistical Society* (Series B), 1961.

(15) Salveson, M. E., "A Problem in Optimal Machine Loading," *Management Science*, April, 1956.

(16) Sandeman, P., "Empirical Design of Priority Waiting Times for Job Shop Control," *Operations Research*, July–August, 1961.

(17) Simon, H. A., "On the Application of Servomechanism Theory in the Study of Production Control," *Econometrica*, April, 1952.

(18) Sisson, R. L., "Sequencing Theory," in *Progress in Operations Research*, R. L. Ackoff, Editor, Volume I, John Wiley and Sons, New York, 1961.