
CHAPTER

SHOP SCHEDULING WITH MANY PRODUCTS

"I mean that part where you say how the First Industrial Revolution devalued muscle work, then the second one devalued routine mental work. . . . That would be the third revolution, I guess—machines that devalue human thinking."

—Kurt Vonnegut, Jr., *Player Piano*

4.1

INTRODUCTION

In the two previous chapters we were concerned with serial or assembly flow systems. All products followed the same route. In addition, production was repetitive with a small number of similar products being manufactured in high volume. In this chapter, we examine systems that manufacture a variety of different products. Products may or may not follow the same route, but relative processing times at workstations vary significantly. Products are released to the production system in **batches** of one or more parts. Batches will also be referred to as jobs. For each job, the production sequence and processing time at each workstation in the sequence are assumed known.

If all batches visit the same sequence of workstations, the system is called a **flow shop**. In a **job shop**, on the other hand, each part type has its own route. These individual routes may be carefully planned by an experienced process planner; however, the total mix of part batches results in a virtually structureless overall material flow pattern.

Because of the variety of product requirements, job shops must be designed for maximum flexibility. Batches must be free to move between any pair of workstations and, normally, to be processed in any order. Individual workstations must be capable of performing a wide variety of tasks. Expertise tends to be process related as opposed to product related, since products may have little or no relationship to one another. Accordingly, job shops tend to be physically and administratively organized by processing function. Milling machines, for instance, form one department and lathes another. A product batch flows between departments, visiting perhaps only one machining center in each department along its route. Throughout most of this book we argue against such configurations. Indeed, perhaps the single most agreed-on principle of manufacturing is that process layouts are less efficient than product

layouts. One widely observed phenomenon is that while in a job shop, jobs spend 95 percent of their time in nonproductive activity. Much of this time is spent waiting in queue. The remaining 5 percent is split between lot setup and processing. Unfortunately, many facilities do not produce high enough volumes of specific parts to justify product layouts. Random batch arrival rates and processing times mean variability in resource requirements through time. If we wish to keep down capital investment and maintain high levels of machine utilization, queueing theory leaves no doubt that long average waiting times for resources will occur. The complexities of scheduling and engineering change orders in this environment exacerbate the problem. Chapter 5 discusses the use of group technology to classify product similarities that will allow medium volume/variety facilities to implement multiple product layouts. However, even this is of little help to the pure custom job shop, where all jobs differ significantly. Customers may demand specific features and practices that prevent exploitation of job similarities. Thus, environments exist in which job shops are necessary.

Throughput time, the time between when a job is released to the shop and when it is completed and ready for delivery, is an important measure for production facilities. From Little's Law we know that WIP inventory is proportional to throughput time with the production rate of jobs being the proportionality constant. Since WIP costs money and occupies space and other resources, we desire just enough WIP to allow workstations to be scheduled effectively. The term "effective" here refers to avoiding machine starvation that results in late delivery of product. Thus our objective should be to find the minimum WIP plan that permits on-time deliveries.

Throughput time is composed of processing time, setup time, move (material handling) time, plus waiting time. Waiting time includes the time spent waiting for machines, waiting to be moved, and waiting for parts that are also needed for the next operation. Processing time per unit is normally fixed by machine capabilities. However, smaller setup times allow smaller economic batch sizes and thus shorter batch processing times. The majority of throughput time tends to be waiting time. Continuous or high-frequency material handling can reduce move wait time. Small batch sizes (high batch processing rates) result in short wait times at machines. In general, actions that reduce the variability in the system allow resources to maintain effective utilization levels without excessive inventory. Variability in workcenter queue lengths, machine availability, and process yield all require that slack time be added to lead time estimates which, in turn, increase WIP levels. Reliable, paced assembly lines do not require inventory between workstations. Even in less repetitive situations, work should be released to the shop to level work loads through time.

Recognizing the importance of throughput time, queueing theory justifies process layouts for a nonrepetitive, highly variable environment. When interarrival and processing times vary, a single group of multiple servers, corresponding to a process layout with machines of a common type grouped together, is more efficient than parallel servers, each with its own queue. The idea of parallel servers with individual queues is similar to having separate serial production lines for different types of products. Another choice may be between general purpose machines, capable of performing many or all of the required operations on a part, and use of a serial line of special purpose machines, each machine intended to perform a subset of the part operations. We will address these issues in more detail in Chapter 11 but, for now, consider the two configurations shown in Figure 4.1. To gain insight into system behavior, we will assume a steady-state environment where jobs arrive randomly (exponential interarrival times) at a rate of 25 per day. Processing times are also

expo
servi
del, s
avera
insten
rial c
rial i
and
five-c
job i
could
lines
ing t
here
four
appe
oper
teran
pern
ever
proc
thro
thro
othe
elim
N
der
conc
duct
even

λ

λ

Figur

exponentially distributed with a mean of 0.1 machine days. With arrival rate $\lambda = 25$, service rate $\mu = 10$, and $c = 4$ servers, Poisson queueing theory (an $M/M/c/\infty$ model, see Table 11.1 for details) tells us that job shop configuration *a* will have an average of 3.0 jobs in process and job throughput time will average 0.12 days. If, instead, we assign one-fourth of the workload to each machine and use the serial configuration *b*, we will see in Chapter 11 that the system behaves like a serial network of $GI/G/1$ queues; we will have 3.5 jobs in the system on average, and throughput times will average 0.14 days. In both instances machines are busy five-eighths of the time, but the use of general machines performing the complete job is clearly preferred. The difference would be much greater if job requirements could not be divided into four equal subtasks. Why then were assembly/transfer lines so useful in the last two chapters? Remember, in those chapters processing times were basically constant as opposed to the exponential assumption used here. If jobs arrived precisely every twenty-fifth of a day and could be divided into four equal subtasks, throughput times would drop to 0.10 days as queueing disappeared in configuration *b*. As utilizations increase, the differences between the operating systems become more exaggerated. Suppose $\lambda = 38$. With constant interarrival and processing times, the serial line still manages fine, since cycle time permits 40 jobs per day. Jobs never wait and throughput time is still 0.1 days. However, suppose the time between job arrivals is exponentially distributed, as is their processing time. The four parallel servers of configuration *a* will yield an average throughput time of 0.55 days. The serial line of configuration *b* will yield an average throughput time of 0.72 days. For repetitive manufacturing, the reader should recall other advantages as well, such as lower skill requirements, faster learning, and setup elimination.

Now that we have seen that job shops have a place in manufacturing, the remainder of this chapter contains some simple deterministic models and basic scheduling concepts. We begin with the issue of when to release individual orders to the production facility. We then look at how to sequence jobs at a single workstation and eventually consider scheduling jobs through the entire facility.

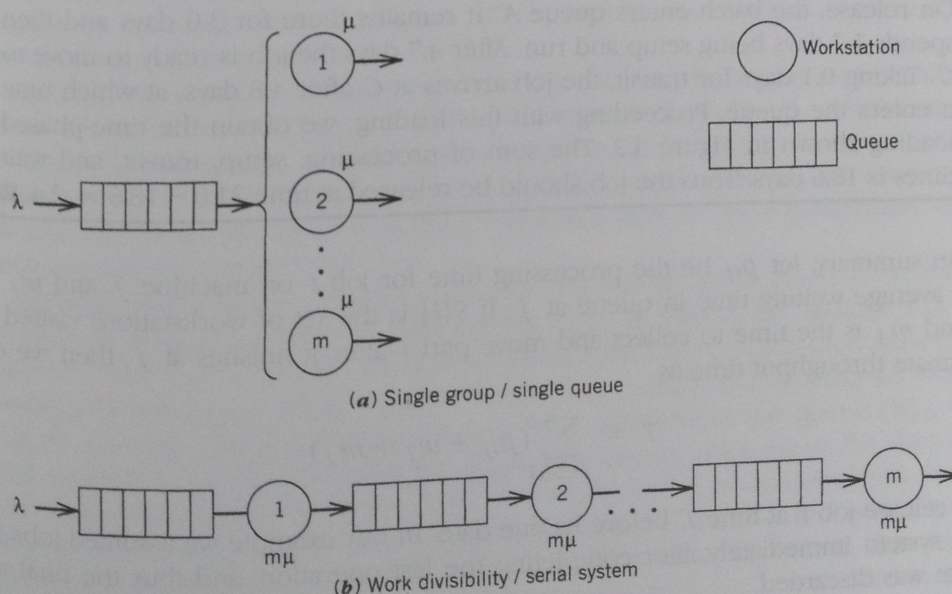


Figure 4.1 Group versus serial organization.

4.2

ORDER RELEASE

Suppose we are assigned the task of releasing jobs to the shop. From a list of pending orders we must select the time to begin processing. The shop manager's objective is to keep all machines busy. The sales department wants to ensure that all customer due dates are met. In either case, it is helpful to know the rate at which released jobs will progress through the shop. The simplest approach is to use average workstation delay times. We will describe this simple approximation by an example.

EXAMPLE 4.1

Consider the product shown in Table 4.1.

Table 4.1 Process Plan for Example 4.1

Operation	Workcenter	Setup Time (days)	Processing Time (days)
1	A	0.5	1.2
2	C	0.1	0.6
3	F	1.0	3.4
4	D	0.3	1.2

Waiting times at workcenters (A, B, C, D, E, F) historically average (3.0, 4.0, 2.5, 1.0, 6.5, 3.5) days, respectively. After completion at a workcenter, jobs usually sit for about 0.1 days before being moved to their next workcenter. A job waiting to be released is due on day 21. When should it be released?

Solution

Using the average times we can project the flow of a batch of this product. On release, the batch enters queue A. It remains there for 3.0 days and then spends 1.7 days being setup and run. After 4.7 days the job is ready to move to C. Taking 0.1 days for transit, the job arrives at C after 4.8 days, at which time it enters the queue. Proceeding with this loading, we obtain the time-phased loading shown in Figure 4.2. The sum of processing, setup, transit, and wait times is 18.6 days. Thus the job should be released at time $21.0 - 18.6 = 2.4$. ■

In summary, let p_{ij} be the processing time for job i on machine j and w_j be the average waiting time in queue at j . If $S\{i\}$ is the set of workstations visited by i and m_j is the time to collect and move part i after it finishes at j , then we can estimate throughput time as

$$T = \sum_{j \in S\{i\}} (p_{ij} + w_j + m_j)$$

We release job i at time T before its due date. In our example we assumed jobs left the system immediately after completing the last operation, and thus the final m_j term was discarded.

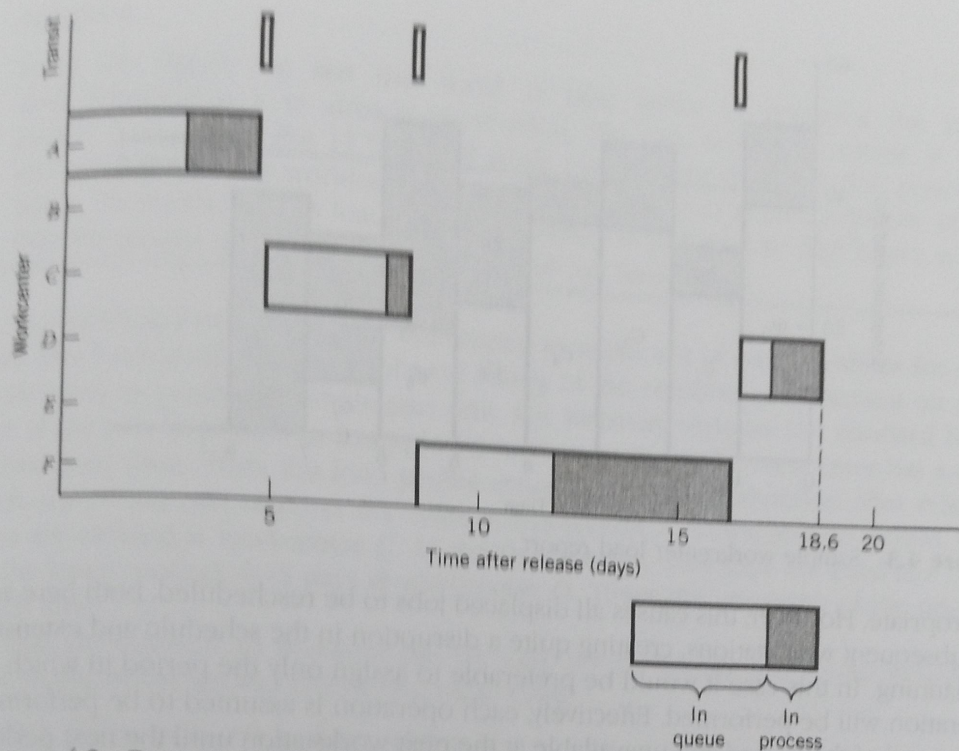


Figure 4.2 Deterministic projection of lead time for Example 4.1.

Unfortunately, this simple approach is valid only under stable conditions without infrequent events of lasting impact. Experience informs us that (1) queues do vary through time and (2) although preventive maintenance is predictable and can be factored into the above scheme, nevertheless, machines do break down at random. Since this is just an average estimate, the prudent manager would release the job earlier to avoid the 50 percent chance that the job takes longer than average and finishes late. The result of such "insurance" is to simply add inventory to the system, which must result in increased waiting times (Little's Law). The increased inventory level causes the variability of queueing time to increase, and hence the manager decides to add even more "insurance." The outcome: output does not increase, but we develop a chaotic mess of excess WIP!

Instead, temporally varying work loads can be partially stabilized by order release strategies that dampen the effects of demand variability. If workstation load predictions are available, dynamic queue averages can be used to improve lead time estimates. Preventive maintenance, process design improvements, and standardized procedures should be used to reduce variability induced by unpredictable events such as machine breakdowns.

Load reports are a common tool for controlling work loads. Finite-loading production planning systems create reports such as Figure 4.3. Assume that it is currently time 0. The report indicates available time (P_t) for each time period t . P_t would reflect any planned maintenance, overtime, or reduction in assigned workers due to schedule changes. Also shown are jobs currently in the queue (W_0), work load scheduled to arrive in period t from jobs already released (E_t). Starting and work load expected to arrive from jobs scheduled to be released with the first operation, the job is tentatively scheduled into the first available time slot after it arrives at the workstation. If a rule other than "first come, first served" (FCFS) is used for selecting the batch to process, the batch can be penciled in where

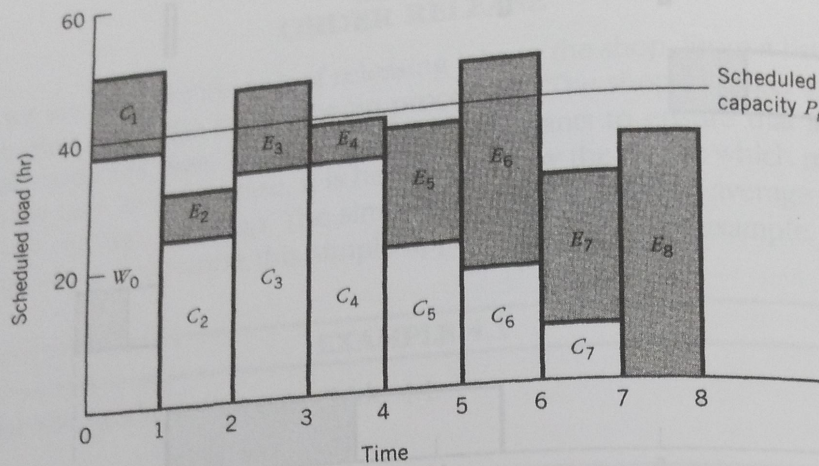


Figure 4.3 Sample workcenter load report.

appropriate. However, this causes all displaced jobs to be rescheduled, both here and at subsequent workstations, creating quite a disruption in the schedule and extensive replanning. In this case it would be preferable to assign only the period in which an operation will be performed. Effectively, each operation is assumed to be performed at the end of the period and unavailable at the next workstation until the next period. In such a system, the minimum flow time for a job is the number of periods equal to its number of operations.

EXAMPLE 4.2

A large job shop is operated with a FCFS rule at workstations. A planner is trying to determine the proper release time for a job requiring 2.5 hours at workstation *L* followed by 6.3 hours at workstation *CG*. All workstations are scheduled for 40 hours of productive time per week. The job is due in 3 weeks. Current load information is shown in Figure 4.4. When should the job be released?

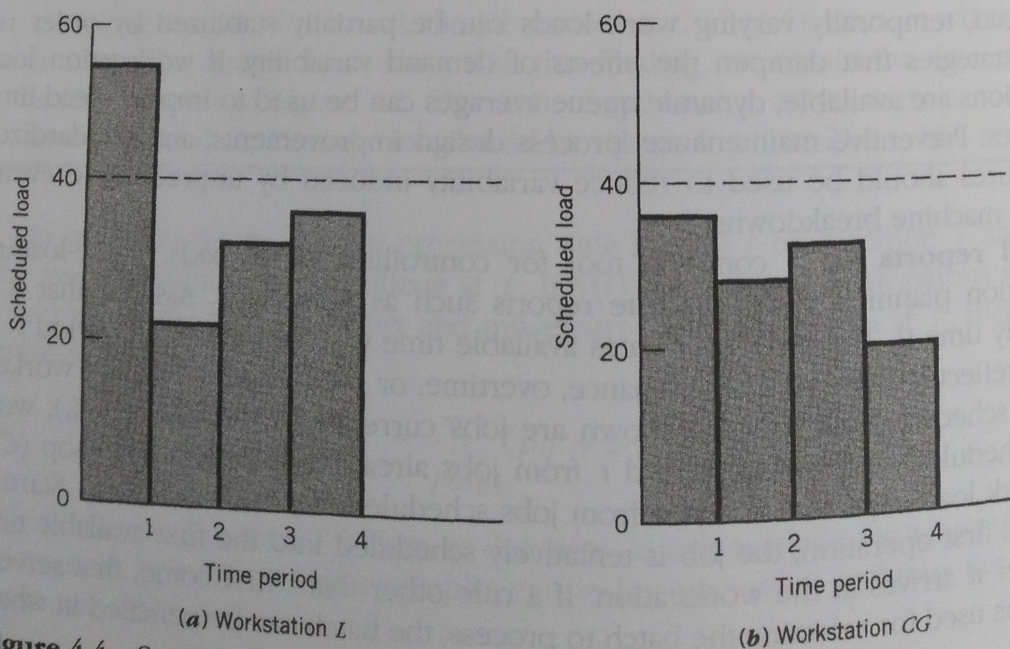


Figure 4.4 Current shop load for Example 4.2.

Solution

From the figure we see that there is little point in releasing the job now. Workstation *L* is already overloaded. We can, however, release it in period 2. Including the 12 hours of work carried over from the first period, this will increase the workload at *L* in the second period to an acceptable 36.5 hours. Workstation *CG*'s load will be increased in period 3 to 38.3 hours, and thus we should be able to complete the job on time. ■

In preparing load reports it is convenient to make use of **load profiles** for each part. The load profile is a time-phased listing of the resource requirement on each workcenter to produce a single part unit. For instance, suppose the standard batch size of the part described in Example 4.1 is 10 units and each workcenter has a one-period lead time. Then the load profile states that in the first period after release, each unit of this part requires 0.17 days at workcenter *A*. In the second period, 0.07 days are needed at workcenter *C*. In the third period, 0.44 days are used at *F*, and in the fourth period, 0.15 days at workcenter *D*. Times are one-tenth of the normal batch time.

Jobs are released so products can be sold, not to occupy space with inventory. Consequently, we should follow two basic rules.

First, if you can't sell it, don't release it!

Second, if you can't make it now, don't release it now!

The rules carry the understanding that normal lead times should be taken into account when asking whether we can make and sell the item. For instance, if the first machine in the production sequence currently has a two-week supply of jobs in its input queue, there is no point in releasing a new job. However, if normal throughput time is two weeks for a part and the two-week queue is at the last machine in its sequence with no other currently released jobs heading for that machine, then it may be time to release the job.

Most manufacturing systems involve assembly of component parts into products prior to shipping. Assembly cannot occur unless all parts are available. In this case, part releases should be coordinated so that the components arrive simultaneously at assembly. Suppose component *A* and component *B* are combined to make a product. Component *A* has a lead time of one week to arrive at assembly, whereas component *B* takes two weeks. Order releases should be offset such that *B* is released a week ahead of *A*. This practice is common in the assembly of complex products such as automobiles. The engine and other subassemblies are fed into the final assembly line with precise timing and synchronization to produce a custom-ordered vehicle. In batch manufacturing, MRP (Materials Requirements Planning) systems attempt to do the same and can be successful if valid part lead times are known. Unfortunately, the attempt to schedule compound assemblies is hindered by the system reliability law encountered in Chapter 1. Suppose parts are released such that each part has a 90 percent chance of being available when it is needed for assembly. If we have 10 parts, then the probability that we can complete the assembly as scheduled is $0.9^{10} = 0.35$. If the assembly has 100 parts, this probability is only 0.00003!

4.2.1 Bottleneck Scheduling

In both job shops and flow shops, for a specified part mix and routings there is a bottleneck process. We define the bottleneck as the workcenter with the highest

utilization, that is, the largest ratio of required processing time to time available.¹ Bottlenecks are easy to identify. By multiplying the load profile by demand rate we obtain the total load on each resource by part type. Summing over part types gives total load per resource. Dividing by time available for each resource we obtain utilization. Thus, utilization for workcenter m is

$$u_m = \frac{\sum_i p_{im} D_i}{P_m}$$

where p_{im} is processing time for part type i at workcenter m , P_m is available (schedule) time for m , and D_i is demand for i . By our definition, the largest utilizations are the bottlenecks. Any workcenter with a utilization greater than one limits sales, frustrating customers, marketing and manufacturing personnel, and stockholders alike. If all workstation utilizations are less than one, then marketing is the bottleneck. It is also possible that material availability may limit production and effectively be the bottleneck.

Scheduling decisions should key on the bottleneck resources as these facilities determine the output rate. Commercial scheduling systems are available that maximize the productive utilization of the bottlenecks and work outward from that point. Predecessor workstations are scheduled to ensure available input into the bottleneck. Orders are released in accord with these schedules. There is simply no reason to release work faster than the bottleneck can process it. Successor workstations are scheduled to handle output of the bottlenecks.

It may be desirable to accumulate significant levels of WIP in front of the bottleneck. Jobs can then be sequenced so as to minimize the time needed to change machines over from one order to the next. Some jobs may be able to share partial or full machine setups. This increases the net processing time available and effectively increases system capacity. Following the objective of keeping the bottleneck busy in productive activity, batch sizes should be large. However, large batch sizes mean long batch processing times and large WIP levels. To avoid this problem while still maintaining efficient usage of the bottleneck, a smaller **transfer batch** can be used. The transfer batch is the number of units transported between workstations. As soon as a number of parts equal to the transfer batch size are completed, the parts are transported to the next workstation. For nonbottleneck stations, the transfer batch can serve as the production batch. Thus, by the time the full production batch is completed at the bottleneck, many of the parts in this batch will have moved downstream and may have already completed subsequent operations. Batch processing times, WIP levels, and throughput times may therefore be small at the nonbottleneck stations.

When the facility produces only a few part types on a repetitive basis, order releases must adhere to the part demand rates. Batch sizes should be determined by standard economic order quantity and setup-time considerations. The basic expression is

$$Q = \sqrt{\frac{2AD}{b}}$$

where Q is the batch size, A is setup cost, D is average demand rate, and b is inventory holding cost per time (generally a percentage of inventory value). In using economic

¹Other definitions of bottleneck occasionally appear. If batch processing times are variable, then a workstation with multiple, parallel servers will have a smaller average queue length than a single server station with the same utilization. Hence, average queue length or time could be used to define the bottleneck.

order quantity workers are capacity, the the workers less than cus to profit and this case.

Han and the best timi compromise demand and in case of a

Sequencing machine. So job order di since we w as soon as has comple **schedule**, called **regu** average and is consider job is made

Before p

Problem

N — the

M — the

d_i — du

p_{ij} — se

Solution

C_i — tin

F_i — the

L_i — late

T_i — tar

We will t time 0, ar are minim to compl

²This assum production efficacy of t

order quantity formulas, setup cost should reflect "the bottom line." For instance, if workers are dedicated to a department that is not a bottleneck and hence has extra capacity, the labor cost component of setup is effectively zero. If not used for setup, the workers are paid to be idle. On the other hand, for a bottleneck with capacity less than customer demand, setup cost should include the cost of lost contribution to profit and overhead while the resource is down for setup. Setups reduce sales in this case.

Han and McGinnis [1988] discuss the use of linear programming for determining the best timing of releases when machines are subject to breakdown. The program compromises between the requisite long-term relative production rates dictated by demand and the desire to keep the bottleneck busy to avoid future resource shortages in case of a breakdown.

4.3

FLOW SHOP SEQUENCING

Sequencing is the process of defining the order in which jobs are to be run on a machine. **Scheduling** is the process of adding start and finish time information to the job order dictated by the sequence. Essentially, the sequence determines the schedule, since we will assume each job is started on a machine as early as possible, that is, as soon as both the job has finished all predecessor operations and the machine has completed all earlier jobs in its sequence.² This is referred to as a **semiactive schedule**, and it can be shown that it is an optimal policy for minimizing what are called **regular measures of performance**. Measures (criteria) in this class include average and maximum completion time, flow time, lateness, and tardiness. A measure is considered "regular" if it is nondecreasing in job completion times, that is, if any job is made to finish later, the measure will stay the same or increase.

Before proceeding, we need a few more definitions.

Problem Variables

N —the number of jobs to be scheduled

M —the number of machines; each job is assumed to visit each machine once

d_i —due date of job i

p_{ij} —setup and processing time of job i on machine j

Solution-Dependent Measures

C_i —time at which job i is completed

F_i —the length of time job i is in the shop (flow time)

L_i —lateness ($C_i - d_i$)

T_i —tardiness ($\max\{0, L_i\}$, i.e., positive lateness values)

We will be assuming that all jobs are in the shop and ready for processing at time 0, and hence flow time and completion time are the same. Typical objectives are minimizing average flow time ($\bar{F} = N^{-1} \sum_i F_i$), minimizing the time required to complete all jobs ($C_{\max}$, also referred to as **makespan**), minimizing average

²This assumption is common in scheduling literature. However, in light of the validity of the just-in-time production philosophy and the recognition that many world-class systems plan for slack capacity, the efficacy of this assumption deserves consideration.

tardiness ($\bar{T} = N^{-1} \sum_i T_i$), minimizing maximum tardiness ($T_{\max}$), and minimizing the number of tardy jobs ($\sum_i \delta_i$, where δ_i is 1 if $T_i > 0$ and 0 otherwise).

Scheduling problems are often denoted by $N/M/A/B$, indicating N jobs, M machines, job flow pattern A , and performance measure B , which is to be appropriately minimized or maximized. Minimizing average flow time ($\bar{F}$) in a general job shop with arbitrary flow pattern (G) would be classified as $N/M/G/\bar{F}$.

In a flow shop, all N jobs must visit machines in the same sequence. It is convenient in this case to number machines in accord with the processing order such that jobs visit machine 1 first and M last. Suppose we also restricted our solution to have the property that all machines process jobs in the same order. This is referred to as a **permutation** schedule, and although the optimal solution for flow shops with more than three machines need not be a permutation schedule, there is usually a permutation schedule that is near optimal.³ Given the sequence, scheduling is easy. At time 0 the first job is started on machine 1. As soon as this operation is completed, the first job begins on machine 2 and the second job begins on machine 1. This process is continued until the last job finishes on machine M . The good news is that we have only to consider $N!$ sequences instead of $(N!)^M$. The bad news is that $N!$ still grows quickly in N .

EXAMPLE 4.3

Consider the set of jobs and processing times shown in Table 4.2. Generate the schedule assuming jobs are processed in the order {2, 4, 1, 3}.

Table 4.2 Flow Shop Processing Times for Example 4.2

Job	Machine			
	1	2	3	4
1	2.0	3.5	1.5	2.0
2	4.5	3.0	2.5	1.0
3	1.5	1.5	5.0	0.5
4	4.0	1.0	2.5	0.5

Solution

The final solution is summarized in the Gantt chart of Figure 4.5. We start by assigning job 2 to machine 1 at time 0. Since $p_{21} = 4.5$, the operation lasts until time 4.5. Since all jobs must go to machine 1 first, the other machines are idle and the other jobs are queued. At time 4.5, job 2 is loaded onto machine 2 and machine 1 starts on job 4, the second job in the sequence. Machine 2 finishes job 2, $p_{22} = 3.0$ time units later (time is now $t = 7.5$). Machine 1 is still busy with job 4; thus, while job 2 is begun on machine 3, machine 2 is idle, waiting for job 4. The remainder of the schedule is shown in the figure. Note that the schedule reflects the rule that machine j starts job i only when job i is finished on machine $j - 1$ and all jobs with earlier locations in the schedule have finished with machine j . ■

³See Theorems 5.1 and 5.2 of French [1982] to see that an optimal permutation schedule exists for minimizing makespan in two- and three-machine flow shops.

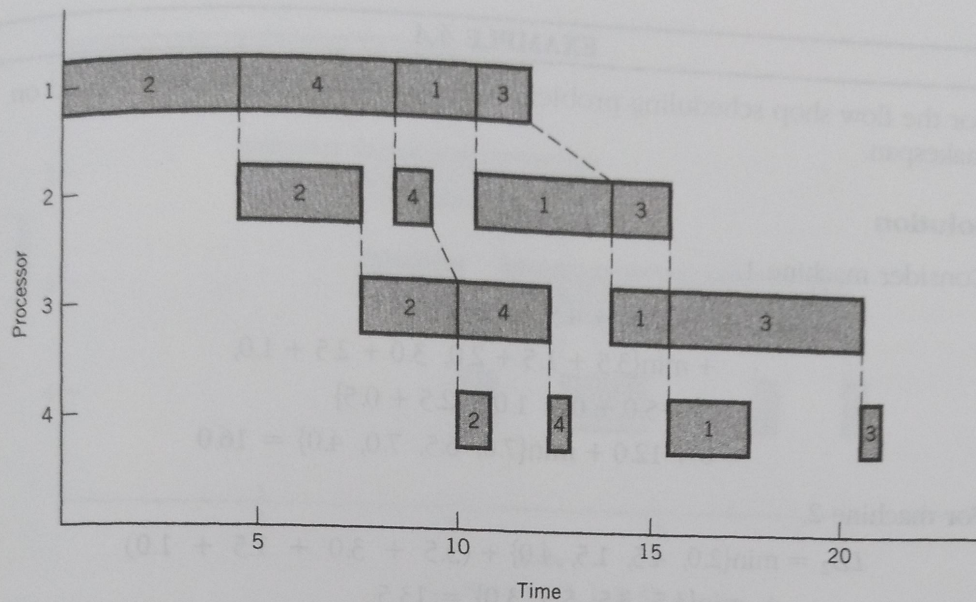


Figure 4.5 Gantt chart for Example 4.3.

The first thing that should strike you about the figure is that even for this relatively well-behaved processing system, machine utilization is not very high. The makespan is 21 time units. During that time the machines are busy 12.0, 9.0, 10.5, and 4.0 time units, respectively. Makespan is a good measure if the facility is truly devoted to this set of jobs during their manufacture. In reality, however, the next schedule of jobs may be assigned to machine 1 starting at any time after 12, and machine 4 could be busy completing the previous schedule until time 10 without conflict. Alternatively, we might have workers assigned to these machines only from the time of their first job arrival until last job completion, time 4.5 to 15.5 for machine 2, instead of the entire 21 time units. Nevertheless, makespan does indicate the length of time for which the facility will be occupied by one or more of the current jobs.

Putting aside for a moment the question of the appropriateness of makespan as a criterion, we may ask whether this is the best makespan schedule. Idle time clearly exists in the schedule, leading us to question its efficiency. It turns out that this is often a difficult question to answer. Nonetheless, we can, at least, try by finding a lower bound on schedule makespan. Each machine supplies a lower bound. The makespan must accommodate (1) the delay before the machine can begin processing, (2) total processing time on the machine, and (3) the remaining processing time for the last job after it leaves the machine. Thus, a lower bound based on machine j is

$$LB_j = \min_i \left\{ \sum_{r=1}^{j-1} p_{ir} \right\} + \sum_{i=1}^N p_{ij} + \min_i \left\{ \sum_{r=j+1}^M p_{ir} \right\} \quad (4.1)$$

Summations from 1 to 0 and $M + 1$ to M are defined to be 0. The three terms in equation 4.1 correspond to a lower bound on when machine j can start, processing time on j , and a lower bound on how soon j must finish before total schedule completion. Since each machine provides a valid lower bound, the largest of these is also a lower bound.

EXAMPLE 4.4

For the flow shop scheduling problem of Example 4.3, find a lower bound on makespan.

Solution

Consider machine 1.

$$\begin{aligned} LB_1 &= 0 + \{2.0 + 4.5 + 1.5 + 4.0\} \\ &\quad + \min\{3.5 + 1.5 + 2.0, 3.0 + 2.5 + 1.0, \\ &\quad 1.5 + 5.0 + 0.5, 1.0 + 2.5 + 0.5\} \\ &= 0 + 12.0 + \min\{7.0, 6.5, 7.0, 4.0\} = 16.0 \end{aligned}$$

For machine 2,

$$\begin{aligned} LB_2 &= \min\{2.0, 4.5, 1.5, 4.0\} + (3.5 + 3.0 + 1.5 + 1.0) \\ &\quad + \min\{3.5, 3.5, 5.5, 3.0\} = 13.5 \end{aligned}$$

For machine 3,

$$\begin{aligned} LB_3 &= \min\{5.5, 7.5, 3.0, 5.0\} + (1.5 + 2.5 + 5.0 + 2.5) \\ &\quad + \min\{2.0, 1.0, 0.5, 0.5\} = 15.0 \end{aligned}$$

Last, for machine 4,

$$LB_4 = \min\{7.0, 10.0, 8.0, 7.5\} + (2.0 + 1.0 + 0.5 + 0.5) + 0 = 11.0$$

The largest (tightest) lower bound is 16.0. We do not know if 16 is obtainable, but we may suspect we can do better than 21. If we examine the LB_1 equation we see that this lower bound came from putting job 4 last. Job 4 went last because it can proceed most quickly through machines 2, 3, and 4. Let's try the schedule (2, 1, 3, 4) formed by putting job 4 last. The Gantt chart is shown in Figure 4.6a. We save a modest 0.5 on makespan. It now appears that the problem with the schedule is that it takes too long to start machines 2 through 4 into operation. Suppose we place job 3 first; it has the smallest time requirement on machine 1 and will let us initiate the other machines sooner. Sequence (3, 2, 1, 4) is shown in Figure 4.6b. Makespan is now 17, just one more than our lower bound of 16. To summarize, we have improved by placing jobs with small p_{i1} early and small $\sum_{j=2}^M p_{ij}$ late in the sequence. Can you see another switch we could make by using these guidelines? Try it and see what happens! ■

4.3.1 Single-Machine Scheduling

Before looking further at the general scheduling problem, let's try some simpler problems. Assume $M = 1$. This may seem like a trivial problem, but there are systems that are either scheduled as a single facility (only one product is manufactured at a time) or for which the schedule of a single bottleneck resource forms the basis for the entire facility.

Several useful results exist for single-machine scheduling. Suppose, for instance, that our objective is minimization of average job flow time. This is equivalent to minimizing average WIP. Since flow times and completion times are the same, we

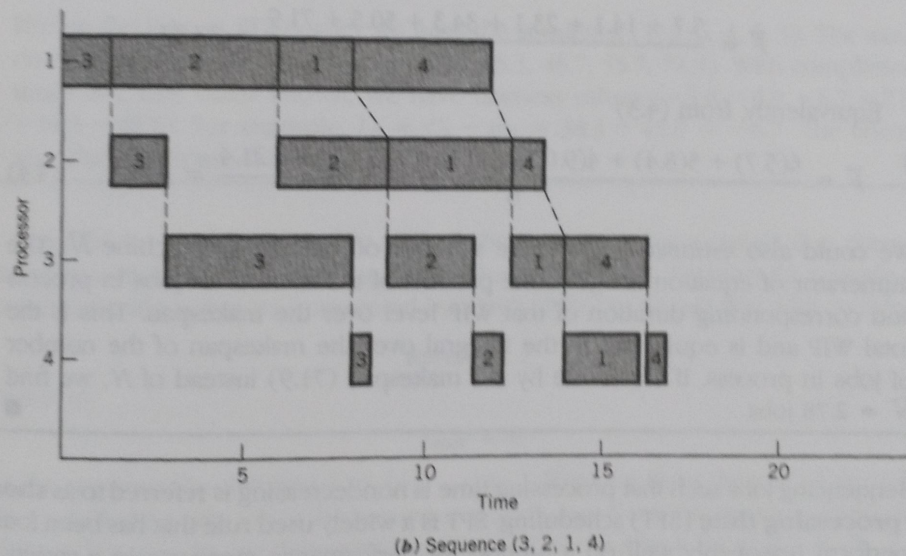
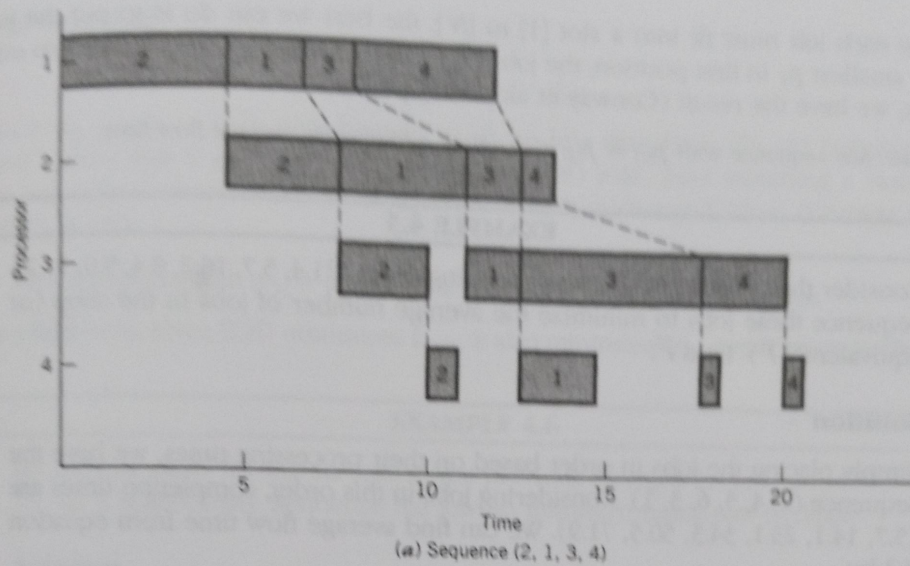


Figure 4.6 Gantt charts for alternate sequences.

can write average flow time as

$$\bar{F} = \frac{\sum_{i=1}^N C_i}{N} \quad (4.2)$$

Let $[i]$ represent the i th ordered job in the production sequence. $C_{[i]} = \sum_{r=1}^i p_{[r]}$. Thus, expression 4.2 becomes

$$\begin{aligned} \bar{F} &= \frac{\sum_{i=1}^N \sum_{r=1}^i p_{[r]}}{N} \\ &= N^{-1} \left[N p_{[1]} + (N-1) p_{[2]} + (N-2) p_{[3]} + \cdots + 1 \cdot p_{[N]} \right] \end{aligned} \quad (4.3)$$

Since each job must fit into a slot [1] to [N], the best we can do is to put the job with smallest p_i in first position, the job with second smallest p_i second, and so on.⁴ Thus, we have the result (Conway et al. [1967], p. 27):

Fact: Any sequence with $p_{[1]} \leq p_{[2]} \leq \dots \leq p_{[N]}$ minimizes average flow time.

EXAMPLE 4.5

Consider the set of six jobs with processing times (21.4, 5.7, 16.2, 8.4, 9.0, 11.2). Sequence these jobs to minimize the average number of jobs in the shop (or equivalently $\bar{F}$). Find $\bar{F}$.

Solution

Simply placing the jobs in order based on their processing times, we have the sequence (2, 4, 5, 6, 3, 1). Considering jobs in this order, completion times are (5.7, 14.1, 23.1, 34.3, 50.5, 71.9). We can find average flow time from equation 4.2 by

$$\bar{F} = \frac{5.7 + 14.1 + 23.1 + 34.3 + 50.5 + 71.9}{6} = 33.27$$

Equivalently, from (4.3)

$$\bar{F} = \frac{6(5.7) + 5(8.4) + 4(9.0) + 3(11.2) + 2(16.2) + 21.4}{6} = 33.27 \quad (4.4)$$

We could also estimate the average number of jobs at the machine $\bar{N}$. The numerator of equation 4.4 gives the product of the number of jobs in process and corresponding duration of that WIP level over the makespan. This is the total WIP and is equivalent to the integral over the makespan of the number of jobs in process. If we divide by the makespan (71.9) instead of N , we find $\bar{N} = 2.78$ jobs. ■

Sequencing jobs such that processing time is nondecreasing is referred to as **shortest processing time** (SPT) scheduling. SPT is a widely used rule that has been found to perform reasonably well on a number of performance measures in a variety of shop environments. Unfortunately, SPT ignores due dates.

As an alternative to SPT, we might sequence jobs in order of their due dates. The job with the **earliest due date** goes first. Earliest due date (EDD) scheduling is also very popular and useful. Suppose we wished to minimize maximum lateness for any job. Let S be a sequence that is not in EDD order. There must be some job $[k]$ in S for which $d_{[k]} > d_{[k+1]}$. Now, $L_{[k]} = C_{[k]} - d_{[k]}$ and $L_{[k+1]} = C_{[k+1]} - d_{[k+1]}$ where $C_{[k]} < C_{[k+1]}$. What happens if we switch jobs $[k]$ and $[k+1]$? The completion times of the other $N - 2$ jobs, and hence their lateness, is unchanged. The relationships between the completion times and due dates for $[k]$ and $[k+1]$ tell us that $L_{[k+1]} > L_{[k]}$ in S . Switching order reduces $C_{[k+1]}$ without affecting $d_{[k+1]}$. Hence, $L_{[k+1]}$ decreases. Unless the new lateness for $[k]$ is larger than the previous lateness for $[k+1]$, we must have at least as good a solution as before. In fact, the new solution is better if $[k+1]$ was the latest job in S . Let's examine the revised lateness for $[k]$.

⁴For the remainder of this single-machine section, we delete the machine subscript and use p_i .

After switching,

$$L_{[k]} = C_{[k+1]} - d_{[k]}$$

Since $d_{[k]} > d_{[k+1]}$, $[k]$'s new lateness must be less than $[k+1]$'s old lateness. We have shown that if any sequence violates the EDD rule, then switching a violating job will either lower maximum lateness or leave it unchanged. Thus, (Conway et al. [1967], p. 31):

Fact: A sequence with $d_{[1]} \leq d_{[2]} \leq \dots \leq d_{[N]}$ minimizes maximum lateness.

As a final note, since EDD minimizes $L_{\max}$, it also minimizes $T_{\max}$, maximum tardiness.

EXAMPLE 4.6

Let the due dates for the six jobs in the previous example be (25., 34., 82., 54., 70., 45.), respectively. Sequence the jobs to minimize maximum tardiness.

Solution

Placing the jobs in EDD order we have the sequence (1, 2, 6, 4, 5, 3). The associated completion times are (21.4, 27.1, 38.3, 46.7, 55.7, 71.9). With completion times and due dates known, we have lateness values (-3.6, -6.9, -6.7, -7.3, -14.3, -10.1). For example, $L_6 = C_6 - d_6 = 38.3 - 45.0 = -6.7$. The corresponding tardiness values are all 0. ■

It would be useful to be able to find the sequence that satisfied due dates but also minimized average flow time among such sequences. Fortunately, a single-pass construction algorithm was presented by Smith [1956] to do just that. Our problem is

$$\text{minimize } \bar{F}$$

subject to:

$$T_{\max} = 0$$

Let the sequence be denoted $S = \{s_{[1]}, \dots, s_{[N]}\}$. Working from the end of the schedule, we will add a new job to the sequence at each step. Consider the task of selecting a job to go last. Due to our delivery constraint, only those jobs k for which $d_k \geq \sum_i p_i$ are admissible, since the last job will have completion time $C_{[N]} = \sum_i p_i$. We know that SPT minimizes $\bar{F}$, and thus long jobs should be at the end of the schedule. This is accomplished by selecting the admissible k with maximum processing time and placing this job last. We now have $N - 1$ jobs to schedule. The process is repeated to select the job that goes $N - 1$ st in sequence. The process continues until all jobs are sequenced. If at some point no admissible jobs are found, then no sequence exists with $T_{\max} = 0$. In this instance our best option is to change the constraint to $T_{\max} = \delta$ for some small δ and try to resolve the problem. The same algorithm is applied by adding δ to all due dates.

Flow Time-Due Date Algorithm

STEP 0. Initialize. Set $k = N + 1$, $A = \{1, \dots, N\}$, $S = \phi$, $\tau = \sum_{i=1}^N p_i$.

STEP 1. Select job. Set $k = k - 1$. Find $s_{[k]} = \arg \max_{i \in A} \{p_i : d_i \geq \tau\}$.

STEP 2. Assign job. Update the list of unassigned jobs $A = A - s_{[k]}$, $\tau = \tau - p_{s_{[k]}}$. If $A = \phi$ stop; otherwise, go to 1.

EXAMPLE 4.7

Consider the single-machine scheduling problem of Table 4.3. The supervisor realizes that all jobs cannot be delivered on time. Is there a sequence for which no job is more than two days late? If so, find it. If options exist, the supervisor would like to minimize the number of jobs at the machine as well.

Table 4.3 Problem Data for Example 4.7

Job	Processing Time (days)	Due Date (day)
1	2.4	11
2	5.0	8
3	4.2	5
4	1.5	12
5	0.9	13

Solution

- STEP 0. Initialize. Total processing time is $\tau = 2.4 + 5.0 + 4.2 + 1.5 + 0.9 = 14.0$. Due dates are adjusted to (13., 10., 7., 14., 15.), respectively, by adding the 2-day allowance.
- STEP 1. Select job. Admissible jobs are those with due dates of 14 or later. This includes jobs 4 and 5. Since $p_4 > p_5$, $s_5 =$ job 4.
- STEP 2. Update. $S = \{.,., 4\}$, $A = \{1, 2, 3, 5\}$, and $\tau = \tau - 1.5 = 12.5$.
- STEP 1. Select job. Admissible jobs with due dates of 12.5 or greater are 1 and 5. Since $p_1 > p_5$, $s_4 = 1$.
- STEP 2. Update. $S = \{.,, 1, 4\}$, $A = \{2, 3, 5\}$, and $\tau = 10.4$.
- STEP 1. Select job. Only job 5 is admissible.
- STEP 2. Update. $S = \{., 5, 1, 4\}$, $A = \{2, 3\}$, and $\tau = 9.5$.
- STEP 1. Select job. Job 2 is now the only admissible job.
- STEP 2. Update. $S = \{2, 5, 1, 4\}$, $A = \{3\}$, $\tau = 4.5$.
- STEP 1. Select job. Job 3 is now admissible.
- STEP 2. Final schedule is $S = \{3, 2, 5, 1, 4\}$.

The final schedule is shown in Figure 4.7. Average flow time is

$$\bar{F} = \frac{4.2 + 9.2 + 10.1 + 12.5 + 14.0}{5} = 10.0$$

Job 2 is 1.2 days late, job 1 is 1.5 days late, and job 4 is 2 days late. As compared with an EDD solution, job 5 was moved up because of its short processing time. The EDD solution $\{3, 2, 1, 4, 5\}$ has $\bar{F} = 10.4$. ■

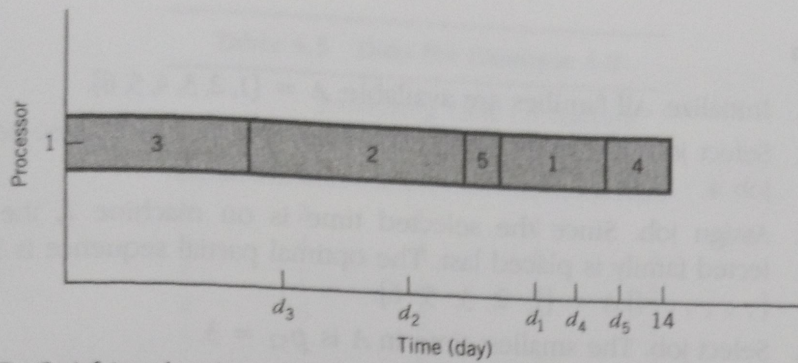


Figure 4.7 Satisfying due-date schedule of Example 4.7.

4.3.2 Two-Machine Flow Shops

Beyond single-machine scheduling, most problems are difficult to solve. However, it is easy to find the minimum makespan schedule for a two-machine flow shop. In fact, we have already encountered the logic for such an algorithm; jobs with short processing times on machine 1 should go early in the schedule and jobs with short processing time on machine 2 should go late in the schedule. We will construct the optimal sequence by adding one job at a time. The selected job goes at either the earliest available slot in the sequence or the latest. Job assignments are permanent. For example, once a job is assigned to be first, it stays in that position and subsequent assignments fall into line behind it.

Johnson's Algorithm

STEP 0. Initialize. $A = \{1, 2, \dots, N\}$.

STEP 1. Select job. If $A = \emptyset$, stop; otherwise, find $k = \arg \min_{k \in A} \{p_{i1} \text{ or } p_{i2}\}$.

STEP 2. Assign job k . If minimum is on machine 1, place k as early as possible in sequence S . If minimum is on machine 2, place k as late as possible in S . $A = A - k$. Go to 1.

EXAMPLE 4.8

A plant produces six families of products. Production consists of parts fabrication followed by assembly. Only one family can be fabricated or assembled at a time. Given the data in Table 4.4, find the minimum makespan schedule for the plant.

Table 4.4 Family Setup and Processing Days for Example 4.8

Family	Parts Fabrication	Assembly
1	6	4
2	10	8
3	4	9
4	7	2
5	6	3
6	5	6

Solution

- STEP 0. Initialize. All families are available: $A = \{1, 2, 3, 4, 5, 6\}$.
- STEP 1. Select job. The minimum processing time is $p_{42} = 2$; so select job 4.
- STEP 2. Assign job. Since the selected time is on machine 2, the selected family is placed last. The optimal partial sequence is $S = \{\ , \ , \ , \ , \ , 4\}$. $A = \{1, 2, 3, 5, 6\}$.
- STEP 1. Select job. The smallest time in A is $p_{52} = 3$.
- STEP 2. Assign job. Family 5 is placed as late as possible, that is, into position 5. $S = \{\ , \ , \ , 5, 4\}$, and $A = \{1, 2, 3, 6\}$.

Continuing in this fashion we select family 1 and place it in position 4 (ties are broken arbitrarily), then we select family 3 for position 1, then family 6 for position 2, and last family 2 goes to position 3. The final sequence is $S = \{3, 6, 2, 1, 5, 4\}$. ■

Johnson's algorithm can be used as a heuristic when $M > 2$. The M machines are split into two pseudo machines. The pseudo machines have processing times equal to the sum of the processing times on the real machines assigned to them. Johnson's algorithm can then be applied to the N -job, two-pseudo-machine problem to obtain a job sequence. The only question is how to split the M machines into the two groups. One well-known scheduling rule is that for the three-machine flow shop, if either

$$\min p_{i1} \geq \max p_{i2} \text{ or } \min p_{i3} \geq \max p_{i2}$$

then the optimal sequence is found from groups $\{1, 2\}$ and $\{2, 3\}$. Thus for the pseudo machines a and b , we use processing times

$$p_{ia} = p_{i1} + p_{i2} \quad p_{ib} = p_{i2} + p_{i3}$$

Campbell et al. [1970] have suggested solving $M - 1$ two-machine problems in general and selecting the best. The $M - 1$ problems come from forming the groups

- $\{1\}, \{M\}$
- $\{1, 2\}, \{M - 1, M\}$
- $\{1, 2, 3\}, \{M - 2, M - 1, M\}$
- $\{1, 2, 3, \dots, M - 1\}, \{2, 3, \dots, M\}$

EXAMPLE 4.9

Consider the four-machine flow problem of Table 4.5. Find a low makespan schedule.

Solution

We must solve three different two-machine flow shop problems. The first problem uses machines 1 and 4 only. The second problem combines processing times on machines 1 with 2 and 3 with 4. The third problem combines 1, 2, and 3 for the first pseudo machine and 2, 3, and 4 for the second. Resulting pseudo processing times are shown in Table 4.6.

Table 4.5 Data for Example 4.9

Job	Processing Time on Machine			
	1	2	3	4
1	2	3	1	5
2	4	2	6	9
3	1	4	8	5
4	6	3	5	2
5	4	2	3	7

Table 4.6 Combined Processing Times for Example 4.9

Job	Machine Combination					
	1	4	1, 2	3, 4	1, 2, 3	2, 3, 4
1	2	5	5	6	6	9
2	4	9	6	15	12	17
3	1	5	5	13	13	17
4	6	2	9	7	14	10
5	4	7	6	10	9	12

The sequences generated by the algorithm for the three problems are $S_1 = \{3, 1, 2, 5, 4\}$, $S_2 = \{1, 3, 2, 5, 4\}$, and $S_3 = \{1, 5, 2, 3, 4\}$. The corresponding schedules are shown in Figure 4.8 on page 114. The best makespan of 34 days results from schedule $S_3 = \{1, 5, 2, 3, 4\}$. Is this schedule optimal? Here's a hint: how soon can machine 4 begin working? ■

4.4

JOB SHOP SCHEDULING

The general job shop problem is to schedule production times for N jobs on M machines. At time 0, we have a set of N jobs. For each job we have knowledge of the sequence of machines required by the job and the processing time on each of those machines. Due dates may also be known. The objective may be to minimize the makespan for completion of all jobs, minimizing the number of tardy jobs or average tardiness, minimizing the average flow time, or achieving some weighted combination of these criteria. This problem is notoriously difficult to solve. On each of the M machines there are $N!$ possible job orderings making a total of $(N!)^M$ possible solutions. For just 10 jobs on five machines that gives over 6×10^{31} choices. Despite considerable effort by researchers, and simplistic abstractions of reality, most scheduling problems are not even close to being solved. Rather than diving into the mathematical quagmire engulfing job shop scheduling, we will skim the top and look at some popular simple rules and some basic results.

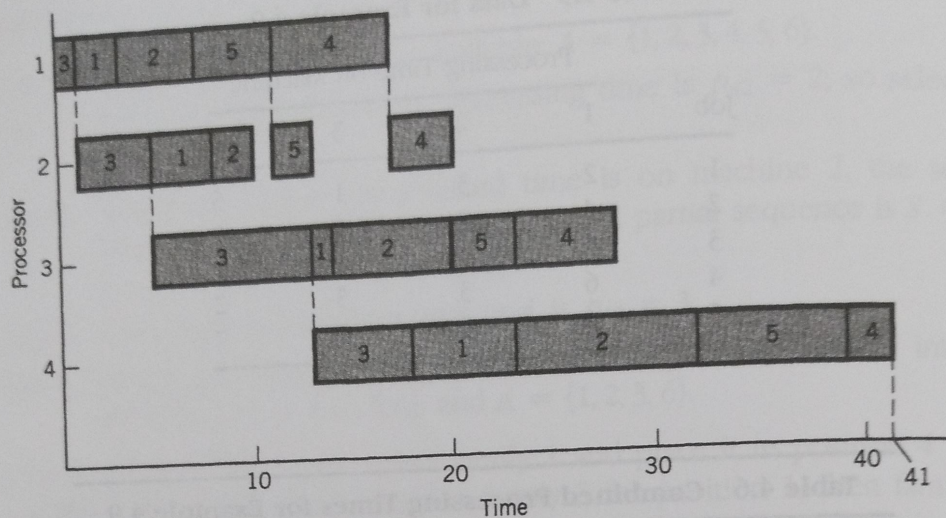
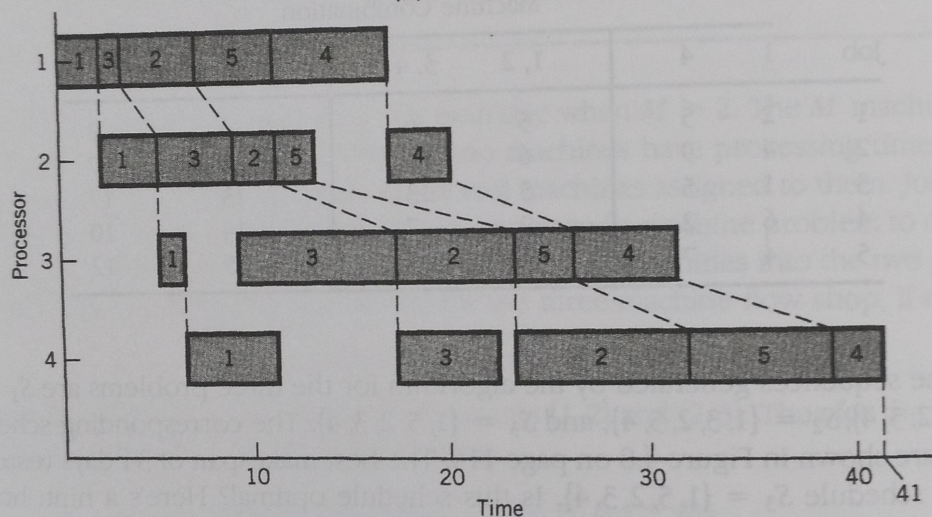
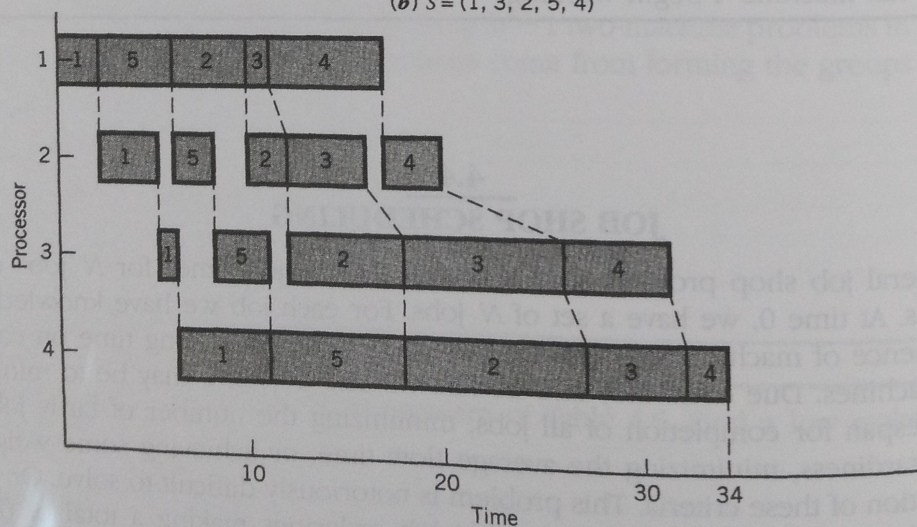
(a) $S = (3, 1, 2, 5, 4)$ (b) $S = (1, 3, 2, 5, 4)$ (c) $S = (1, 5, 2, 3, 4)$

Figure 4.8 Gantt charts for Example 4.9 candidate V sequences.

4.4.1 Dispatching Rules

When a processor becomes available, a job must be selected from its input queue for immediate setup and processing. We refer to this as **dispatching**. Simple dispatching rules are often used in shop scheduling. Table 4.7 lists a dozen of the more popular rules. Rules can be classified as being **static** or **dynamic**. Static rules have priority indices that stay constant as jobs travel through the plant, whereas dynamic rules change with time and queue characteristics. LTWK and EDD (assuming due dates are fixed) are static rules. LWKR is dynamic, since the remaining processing time decreases as the job progresses through the shop, that is, through time. **Slack-based** rules are also dynamic. Slack is defined as $\text{slack} = \text{due date} - \text{current time} - \text{remaining work}$, that is, the time until the due date minus remaining processing time. Due dates may also change while the job order is open; however, Berry et al. [1984] concluded that for reorder point inventory systems, use of static due dates based on average demand rates is preferable to the use of dynamic due dates, which constantly reflect random fluctuations in demand. Rules can also be classified as **myopic** or **global**. Myopic rules look only at the individual machine, whereas global rules look at the entire shop. SPT is myopic but WINQ is global.

Table 4.7 Standard Dispatching Rules

Name	Description
SPT	(Shortest Processing Time) Select a job with minimum processing time.
EDD	(Earliest Due Date) Select a job due first.
FCFS	(First Come, First Served) Select a job that has been in the workstation's queue the longest.
FISFS	(First in System, First Served) Select a job that has been on the shop floor the longest.
S/RO	(Slack per Remaining Operation) Select a job with the smallest ratio of slack to operations remaining to be performed.
Covert	Order jobs based on ratio of slack-based priority to processing time.
LTWK	(Least Total Work) Select a job with smallest total processing time.
LWKR	(Least Work Remaining) Select a job with smallest total processing time for unfinished operations.
MOPNR	(Most Operations Remaining) Select a job with the most operations remaining in its processing sequence.
MWKR	(Most Work Remaining) Select a job with the most total processing time remaining.
RANDOM	(Random) Select a job at random.
WINQ	(Work in Next Queue) Select a job whose subsequent machine currently has the shortest queue.

EXAMPLE 4.10

Current time is 10. Machine *B* has just finished a job, and it is time to select its next job. Table 4.8 provides information on the four jobs available. For each of the dispatching rules above, determine the corresponding sequence.

LWKR: Remaining processing time until job completion is (12, 5, 2, 8). The corresponding LWKR sequence is {3, 2, 4, 1}.

MOPNR: The number of remaining operations by job is (3, 2, 1, 2). The MOPNR sequence is {1, 2, 4, 3}.

MWKR: Remaining work loads were found in LWKR. Reversing that order we have sequence {1, 4, 2, 3}.

RANDOM: Using a random number table the author obtained the sequence {3, 1, 2, 4}.

WINQ: We require information on the length of the queue at machines A and C. Suppose these values are 10 and 4. Job 3 goes first since it has no next queue. Jobs 2 and 4 come next since they are headed for machine C, which has less work in its queue than A. Our sequence is thus {3, 2, 4, 1}. ■

4.4.2 Schedule Generation

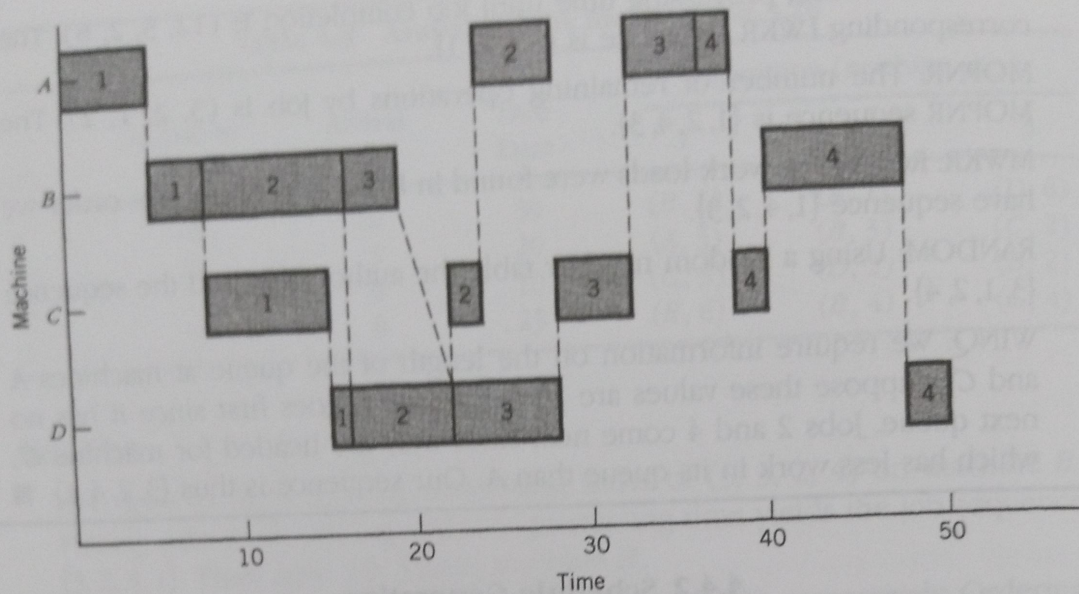
We have already argued the futility of trying to generate optimal schedules in a random environment. Nevertheless, techniques have been developed using optimization ideas, such as dynamic programming and branch and bound, to do just that. We will not discuss these techniques, but it is useful to consider what is known about the optimal schedule. Semiactive schedules were defined earlier. Not all semiactive schedules must be examined. The optimal schedule must be **fully active**. Active schedules are those that never make a job wait in queue when it can be completely processed before the next job is scheduled to start. Active scheduling recognizes that by going ahead and producing the job, nothing is delayed. Time that would otherwise be wasted is used productively. Consider the four-machine, four-job scheduling problem shown in Table 4.9.

Table 4.9 Sample Scheduling Problem

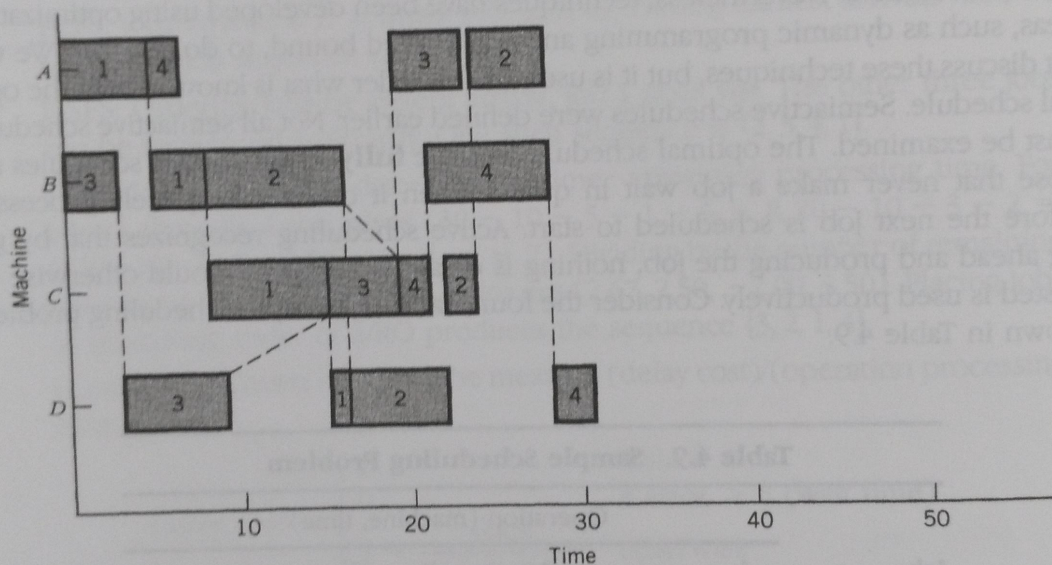
Job	Operation (machine, time)			
	1	2	3	4
1	(A, 5)	(B, 3)	(C, 7)	(D, 1)
2	(B, 8)	(D, 6)	(C, 2)	(A, 5)
3	(B, 3)	(D, 6)	(C, 4)	(A, 4)
4	(A, 2)	(C, 2)	(B, 7)	(D, 3)

Job 1 has high priority, and it has been decided that this job should not be delayed if possible. Adhering to this request, consider the semiactive permutation schedule {1, 2, 3, 4}. This schedule is shown in Figure 4.9a. Adhering to the technological ordering of each job's operations and the permutation requirement for each machine, every operation is scheduled as early as possible. The permutation schedule is not active. Job 3 could be started at time 0 on machine B without delaying job 1. Job 4 could also start on machine A before jobs 2 and 3 are ready for machine A. Moving job 3 forward first and then job 4, we obtain the active schedule shown in Figure 4.9b. Note the significant reduction in makespan.

The next step in schedule elimination is to consider only **nondelay** schedules. Nondelay schedules are such that a machine is never idle when its queue



(a) Permutation schedule (1, 2, 3, 4)



(b) Active schedule

Figure 4.9 Four-job, four-machine schedules.

is nonempty. In active scheduling we started a job if it could be completed before the next job was scheduled to start. In nondelay scheduling we start a job if it can be started before the next job is scheduled to start. This is a logical policy but not necessarily optimal. It may have been preferable to wait for a specific job that is important to the grand scheme. The active schedule of Figure 4.9b is not a nondelay schedule. Job 4 could begin on machine C at time 7. However, it cannot finish on machine C until after the currently scheduled start time for job 1. Nevertheless, a nondelay rule would require job 4 to be processed on C from time 7 to 9, thus delaying job 1. This move may be undesirable if job 1 truly dominates in importance.

The process of generating all the nondelay schedules (or the active schedules for that matter) is straightforward. The number of such schedules can be computationally prohibitive, however. We will present a procedure that will al-

low generation of as many of the nondelay schedules as desired. Basically, we construct a schedule by scheduling one operation at a time. At any point in time each job is either finished or has one operation that can be scheduled. We will simply move forward in time and whenever one or more operations is ready to be scheduled and its desired machine is available, we schedule an operation. If we select one of the available operations at each point and continue until all MN operations are assigned, we have one complete nondelay schedule. If instead we form a tree of all possible decisions at each decision point, all possible nondelay schedules are formed.

At stage t , let

S_t be the partial schedule of $(t - 1)$ scheduled operations.

A_t be the set of operations schedulable at stage t , that is, all predecessor operations are in S_t .

e_k be the earliest time that operation $k \in A_t$ can be scheduled, that is, predecessors are completed and machine is available. Then:

Nondelay Schedule Generation

STEP 0. Initialize. Let $t = 1, S_1 = \phi$. A_1 contains the first operation of each ready job.

STEP 1. Select operation. Find $e^* = \min_{k \in A_t} e_k$. If several e^* exist, choose arbitrarily. Let m^* be the machine needed by e^* . Choose any $k \in A_t$ that requires m^* and has $e_k = e^*$. (If all nondelay schedules are to be created, choose all such k and create a new partial schedule for each.)

STEP 2. Increment. Add the selected operation k to S_t to create S_{t+1} . Remove k from A_t and add the next operation for its job unless that job is completed; this creates A_{t+1} . Set $t = t + 1$. If $t = MN$ stop; otherwise go to 1.

An example will illustrate the procedure. Let o_{ir} denote the r th operation of job i .

EXAMPLE 4.11

Generate a nondelay schedule for the four-job, four-machine problem data of Table 4.9. When choices between operations exist, give highest priority to job 1. Otherwise, choose based on SPT.

Solution

STEP 0. Initialize. $t = 1, S_1 = \phi$, and $A_1 = \{o_{11}, o_{21}, o_{31}, o_{41}\}$.

STEP 1. Select operation. The early start time of all first operations is 0. Job 1 has priority, so select operation o_{11} , which implies $m^* = A$. Operations o_{11} and o_{41} are eligible for selection. We will choose o_{11} due to job 1's priority. (If all nondelay schedules were to be generated, we would have to start two different schedules at this point and continue each through 16 operation assignments.)

STEP 2. Increment. $t = 2, S_2 = \{o_{11}\}, A_2 = \{o_{12}, o_{21}, o_{31}, o_{41}\}$. Early start times for the four operations in A_2 are now (5, 0, 0, 5). Note that operation o_{41} has been moved back since we decided to place operation o_{11} ahead of it.

STEP 1. Select operation. We must choose between operations o_{21} and o_{31} , since they have the earliest start time. Choose o_{21} , which yields $m^* = B$. Both operations are eligible since both require B . Neither operation is from job 1. Invoking the SPT tie-breaker, we choose o_{31} and place this job first on machine B .

The process continues until all 16 operations are assigned. Steps are summarized in Table 4.10, and the final schedule is shown in Figure 4.10. In each row of the table, m^* was selected to be one of the machines used by an element of A_t with the minimum earliest start time. We have a new best makespan of 28. ■

Table 4.10 Construction of Nondelay Schedule for Example 4.11

Step t	A_t	e_k	m^*	k
1	$O_{11}, O_{21}, O_{31}, O_{41}$	0, 0, 0, 0	A	O_{11}
2	$O_{12}, O_{21}, O_{31}, O_{41}$	5, 0, 0, 5	B	O_{31}
3	$O_{12}, O_{21}, O_{32}, O_{41}$	5, 3, 3, 5	B	O_{21}
4	$O_{12}, O_{22}, O_{32}, O_{41}$	11, 11, 3, 5	D	O_{32}
5	$O_{12}, O_{22}, O_{33}, O_{41}$	11, 11, 9, 5	A	O_{41}
6	$O_{12}, O_{22}, O_{33}, O_{42}$	11, 11, 9, 7	C	O_{42}
7	$O_{12}, O_{22}, O_{33}, O_{43}$	11, 11, 9, 11	C	O_{33}
8	$O_{12}, O_{22}, O_{34}, O_{43}$	11, 11, 13, 11	B	O_{12}
9	$O_{13}, O_{22}, O_{34}, O_{43}$	14, 11, 13, 14	D	O_{22}
10	$O_{13}, O_{23}, O_{34}, O_{43}$	14, 17, 13, 14	C	O_{34}
11	$O_{13}, O_{23}, -, O_{43}$	14, 17, -, 14	C	O_{13}
12	$O_{14}, O_{23}, -, O_{43}$	21, 21, -, 14	B	O_{43}
13	$O_{14}, O_{23}, -, O_{44}$	21, 21, -, 21	D	O_{14}
14	$-, O_{23}, -, O_{44}$	-, 21, -, 22	C	O_{23}
15	$-, O_{24}, -, O_{44}$	-, 23, -, 22	D	O_{44}
16	$-, O_{24}, -, -$	-, 23, -, -	A	O_{24}

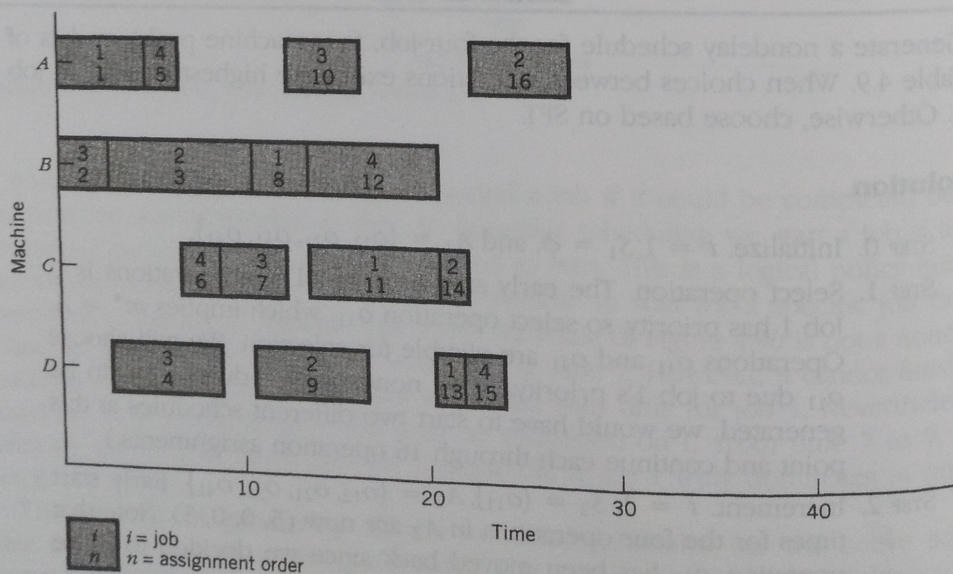


Figure 4.10 Nondelay schedule generated in Example 4.11.

4.5 SUMMARY

Job shops appear inefficient largely because of the production demands placed on them. Order release strategies should be carefully planned to meet due dates while attempting to smooth resource utilization through time.

A few scheduling problems such as minimizing mean flow time or maximum lateness for a single machine are easy to solve. However, general problems such as minimizing average tardiness or scheduling in dynamic, multimachine job shops are notoriously difficult. Even most oversimplified academic problems, which assume perfect knowledge and a deterministic world, are hard. The dynamic nature of actual shops requires real-time schedule updating. For this reason, heuristic dispatching rules are widely used in job shops. SPT performs well in minimizing average flow time and normally makespan as well. EDD minimizes maximum lateness or tardiness.

Permutation schedules work well in flow shops but not necessarily in job shops. If a job has increasing processing times from its first to last operation, it should most likely be early in the production schedule of machines. Likewise, jobs with decreasing processing times should be scheduled later.

For regular measures of performance, there is an active schedule that is optimal. The best nondelay schedule may not be optimal but will be close in most cases. Nondelay schedules also reduce the number of schedules that must be examined and follow the oft-imposed organizational restriction that machines not sit idle while work sits in their input queue.

McKay et al. [1988] defined the real scheduling problem as "how to schedule and dispatch work in such a way that many stated and unstated conflicting goals are satisfied using hard and soft information that is possibly incomplete, biased, outdated and erroneous." Expert systems (Fox and Smith [1984]) have the ability to capture experience and make decisions based on complex interactions of current system status data and objectives. As such, they may be potentially useful in complex scheduling environments.

REFERENCES

- Baker, Kenneth R. (1974), *Introduction to Sequencing and Scheduling*, John Wiley & Sons, New York.
- Berry, William L., Richard J. Penlesky, and Thomas E. Vollmann (1984), "Critical Ratio Scheduling: Dynamic Due-Date Procedures under Demand Uncertainty," *IIE Transactions*, 16(1), 81-89.
- Blackstone, John H. Jr., Don T. Phillips, and Gary L. Hogg (1982), "A State of the Art Survey of Dispatching Rules for Manufacturing Job Shop Operation," *International Journal of Production Research*, 20(1), 27-45.
- Campbell, Herbert G., Richard A. Dudek, and Milton L. Smith (1970), "A Heuristic for the N Job, M Machine Sequencing Problem," *Management Science*, 16(10), 630-637.
- Conway, Richard W., William L. Maxwell, and Louis W. Miller (1967), *Theory of Scheduling*, Addison-Wesley, Reading, Mass.
- Fox, Mark S., and Stephen E. Smith (1984), "ISIS—A Knowledge-Based System for Factory Scheduling," *Expert Systems*, 1(1), 25-49.
- French, Simon (1982), *Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop*, John Wiley & Sons, New York.
- Goldratt, Eliyahu M., and Jeff Cox (1984), *The Goal: Excellence in Manufacturing*, North River Press, New York.
- Han, Min-Hong, and Leon F. McGinnis (1988), "Throughput Rate Maximization in Flexible Manufacturing Cells," *IIE Transactions*, 20(4), 409-417.
- Hopp, Wallace J., Mark L. Spearman, and David L. Woodruff (1990), "Practical Strategies for Lead Time Reduction," *Manufacturing Review*, 3(2), 78-84.
- Johnson, S. M. (1954), "Optimal Two and Three Stage Production Schedules with Setup Times