Chapter 2

Deterministic Models: Preliminaries

Over the last fifty years a considerable amount of research effort has been focused on deterministic scheduling. The number and variety of models considered is astounding. During this time a notation has evolved that succinctly captures the structure of many (but for sure not all) deterministic models that have been considered in the literature.

The first section in this chapter presents an adapted version of this notation. The second section contains a number of examples and describes some of the shortcomings of the framework and notation. The third section describes several classes of schedules. A class of schedules is typically characterized by the freedom the scheduler has in the decision-making process. The last section discusses the complexity of the scheduling problems introduced in the first section. This last section can be used, together with Appendixes D and E, to classify scheduling problems according to their complexity.

2.1 Framework and Notation

In all the scheduling problems considered the number of jobs and the number of machines are assumed to be finite. The number of jobs is denoted by n and the number of machines by m. Usually, the subscript j refers to a job while the subscript i refers to a machine. If a job requires a number of processing steps or operations, then the pair (i, j) refers to the processing step or operation of job j on machine i. The following pieces of data are associated with job j.

Processing time (p_{ij}) The p_{ij} represents the processing time of job j on machine i. The subscript i is omitted if the processing time of job j does not depend on the machine or if job j is only to be processed on one given machine.

Release date (r_j) The release date r_j of job j may also be referred to as the ready date. It is the time the job arrives at the system, i.e., the earliest time at which job j can start its processing.

Due date (d_j) The due date d_j of job j represents the committed shipping or completion date (i.e., the date the job is promised to the customer). Completion of a job after its due date *is* allowed, but then a penalty is incurred. When a due date *must* be met it is referred to as a *deadline* and denoted by $\bar{d}_j$.

Weight (w_j) The weight w_j of job j is basically a priority factor, denoting the importance of job j relative to the other jobs in the system. For example, this weight may represent the actual cost of keeping the job in the system. This cost could be a holding or inventory cost; it also could represent the amount of value already added to the job.

A scheduling problem is described by a triplet $\alpha \mid \beta \mid \gamma$. The α field describes the machine environment and contains just one entry. The β field provides details of processing characteristics and constraints and may contain no entry at all, a single entry, or multiple entries. The γ field describes the objective to be minimized and often contains a single entry.

The possible machine environments specified in the α field are:

Single machine (1) The case of a single machine is the simplest of all possible machine environments and is a special case of all other more complicated machine environments.

Identical machines in parallel (Pm) There are m identical machines in parallel. Job j requires a single operation and may be processed on any one of the m machines or on any one that belongs to a given subset. If job j cannot be processed on just any machine, but only on any one belonging to a specific subset M_j, then the entry M_j appears in the β field.

Machines in parallel with different speeds (Qm) There are m machines in parallel with different speeds. The speed of machine i is denoted by v_i. The time p_{ij} that job j spends on machine i is equal to p_j/v_i (assuming job j receives all its processing from machine i). This environment is referred to as *uniform* machines. If all machines have the same speed, i.e., $v_i = 1$ for all i and $p_{ij} = p_j$, then the environment is identical to the previous one.

Unrelated machines in parallel (Rm) This environment is a further generalization of the previous one. There are m different machines in parallel. Machine i can process job j at speed v_{ij}. The time p_{ij} that job j spends on machine i is equal to p_j/v_{ij} (again assuming job j receives all its processing from machine i). If the speeds of the machines are independent of the jobs, i.e., $v_{ij} = v_i$ for all i and j, then the environment is identical to the previous one.

Flow shop (Fm) There are m machines in series. Each job has to be processed on each one of the m machines. All jobs have to follow the same route, i.e., they have to be processed first on machine 1, then on machine 2, and so on. After completion on one machine a job joins the queue at the next machine. Usually, all queues are assumed to operate under the *First In First Out (FIFO)* discipline, that is, a job cannot "pass" another while waiting in a queue. If the *FIFO* discipline is in effect the flow shop is referred to as a *permutation* flow shop and the β field includes the entry *prmu*.

Flexible flow shop (FFc) A flexible flow shop is a generalization of the flow shop and the parallel machine environments. Instead of m machines in series there are c stages in series with at each stage a number of identical machines in parallel. Each job has to be processed first at stage 1, then at stage 2, and so on. A stage functions as a bank of parallel machines; at each stage job j requires processing on only one machine and any machine can do. The queues between the various stages may or may not operate according to the *First Come First Served (FCFS)* discipline. (Flexible flow shops have in the literature at times also been referred to as hybrid flow shops and as multi-processor flow shops.)

Job shop (Jm) In a job shop with m machines each job has its own predetermined route to follow. A distinction is made between job shops in which each job visits each machine at most once and job shops in which a job may visit each machine more than once. In the latter case the β-field contains the entry *rcrc* for *recirculation.*

Flexible job shop (FJc) A flexible job shop is a generalization of the job shop and the parallel machine environments. Instead of m machines in series there are c work centers with at each work center a number of identical machines in parallel. Each job has its own route to follow through the shop; job j requires processing at each work center on only one machine and any machine can do. If a job on its route through the shop may visit a work center more than once, then the β-field contains the entry *rcrc* for recirculation.

Open shop (Om) There are m machines. Each job has to be processed again on each one of the m machines. However, some of these processing times may be zero. There are no restrictions with regard to the routing of each job through the machine environment. The scheduler is allowed to determine a route for each job and different jobs may have different routes.

The processing restrictions and constraints specified in the β field may include multiple entries. Possible entries in the β field are:

Release dates (r_j) If this symbol appears in the β field, then job j cannot start its processing before its release date r_j. If r_j does not appear in the β field, the processing of job j may start at any time. In contrast to release dates, due dates are not specified in this field. The type of objective function gives sufficient indication whether or not there are due dates.

Preemptions (*prmp*) Preemptions imply that it is not necessary to keep a job on a machine, once started, until its completion. The scheduler is allowed to interrupt the processing of a job (preempt) at any point in time and put a different job on the machine instead. The amount of processing a preempted job already has received is not lost. When a preempted job is afterwards put back on the machine (or on another machine in the case of parallel machines), it only needs the machine for its *remaining* processing time. When preemptions are allowed *prmp* is included in the β field; when *prmp* is not included, preemptions are not allowed.

Precedence constraints (*prec*) Precedence constraints may appear in a single machine or in a parallel machine environment, requiring that one or more jobs may have to be completed before another job is allowed to start its processing. There are several special forms of precedence constraints: if each job has at most one predecessor and at most one successor, the constraints are referred to as *chains*. If each job has at most one successor, the constraints are referred to as an *intree*. If each job has at most one predecessor the constraints are referred to as an *outtree*. If no *prec* appears in the β field, the jobs are not subject to precedence constraints.

Sequence dependent setup times (s_{jk}) The s_{jk} represents the sequence dependent setup time that is incurred between the processing of jobs j and k; s_{0k} denotes the setup time for job k if job k is first in the sequence and s_{j0} the clean-up time after job j if job j is last in the sequence (of course, s_{0k} and s_{j0} may be zero). If the setup time between jobs j and k depends on the machine, then the subscript i is included, i.e., s_{ijk}. If no s_{jk} appears in the β field, all setup times are assumed to be 0 or sequence independent, in which case they are simply included in the processing times.

Job families (*fmls*) The n jobs belong in this case to F different job families. Jobs from the same family may have different processing times, but they can be processed on a machine one after another without requiring any setup in between. However, if the machine switches over from one family to another, say from family g to family h, then a setup is required. If this setup time depends on both families g and h and is sequence dependent, then it is denoted by s_{gh}. If this setup time depends only on the family about to start, i.e., family h, then it is denoted by s_h. If it does not depend on either family, it is denoted by s.

Batch processing (*batch*(b)) A machine may be able to process a number of jobs, say b, simultaneously; that is, it can process a batch of up to b jobs at the same time. The processing times of the jobs in a batch may not be all the same and the entire batch is finished only when the last job of the batch has been completed, implying that the completion time of the entire batch is determined by the job with the longest processing time. If $b = 1$, then the problem reduces to a conventional scheduling environment. Another special case that is of interest is $b = \infty$, i.e., there is no limit on the number of jobs the machine can handle at any time.

Breakdowns (*brkdwn*) Machine breakdowns imply that a machine may not be continuously available. The periods that a machine is not available are, in this part of the book, assumed to be fixed (e.g., due to shifts or scheduled maintenance). If there are a number of identical machines in parallel, the number of machines available at any point in time is a function of time, i.e., $m(t)$. Machine breakdowns are at times also referred to as machine availability constraints.

Machine eligibility restrictions (M_j) The M_j symbol may appear in the β field when the machine environment is m machines in parallel (Pm). When the M_j is present, not all m machines are capable of processing job j. Set M_j denotes the set of machines that can process job j. If the β field does not contain M_j, job j may be processed on any one of the m machines.

Permutation (*prmu*) A constraint that may appear in the flow shop environment is that the queues in front of each machine operate according to the *First In First Out (FIFO)* discipline. This implies that the order (or *permutation*) in which the jobs go through the first machine is maintained throughout the system.

Blocking (*block*) Blocking is a phenomenon that may occur in flow shops. If a flow shop has a limited buffer in between two successive machines, then it may happen that when the buffer is full the upstream machine is not allowed to release a completed job. Blocking implies that the completed job has to remain on the upstream machine preventing (i.e., blocking) that machine from working on the next job. The most common occurrence of blocking that is considered in this book is the case with zero buffers in between any two successive machines. In this case a job that has completed its processing on a given machine cannot leave the machine if the preceding job has not yet completed its processing on the next machine; thus, the blocked job also prevents (or blocks) the next job from starting its processing on the given machine. In the models with blocking that are considered in subsequent chapters the assumption is made that the machines operate according to *FIFO*. That is, *block* implies *prmu*.

No-wait (*nwt*) The *no-wait* requirement is another phenomenon that may occur in flow shops. Jobs are not allowed to wait between two successive machines. This implies that the starting time of a job at the first machine has to be delayed to ensure that the job can go through the flow shop without having to wait for any machine. An example of such an operation is a steel rolling mill in which a slab of steel is not allowed to wait as it would cool off during a wait. It is clear that under *no-wait* the machines also operate according to the *FIFO* discipline.

Recirculation (*rcrc*) Recirculation may occur in a job shop or flexible job shop when a job may visit a machine or work center more than once.

Any other entry that may appear in the β field is self explanatory. For example, $p_j = p$ implies that all processing times are equal and $d_j = d$ implies that all due dates are equal. As stated before, due dates, in contrast to release dates, are usually not explicitly specified in this field; the type of objective function gives sufficient indication whether or not the jobs have due dates.