Michael L. Pinedo

Scheduling

Theory, Algorithms, and Systems

Fifth Edition

EXTRAS ONLINE

Springer

Scheduling

Michael L. Pinedo

Scheduling

Theory, Algorithms, and Systems

Fifth Edition

Springer

Michael L. Pinedo
IOMS Dept Rm 8-59 KMC
NYU Stern School of Business
New York, NY, USA

Additional material to this book can be downloaded from http://extras.springer.com

ISBN 978-3-319-26578-0   ISBN 978-3-319-26580-3 (eBook)
DOI 10.1007/978-3-319-26580-3

Library of Congress Control Number: 2015955728

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media (www.springer.com)

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.

M.L. Pinedo, *Scheduling*, DOI 10.1007/978-3-319-26580-3_2

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.

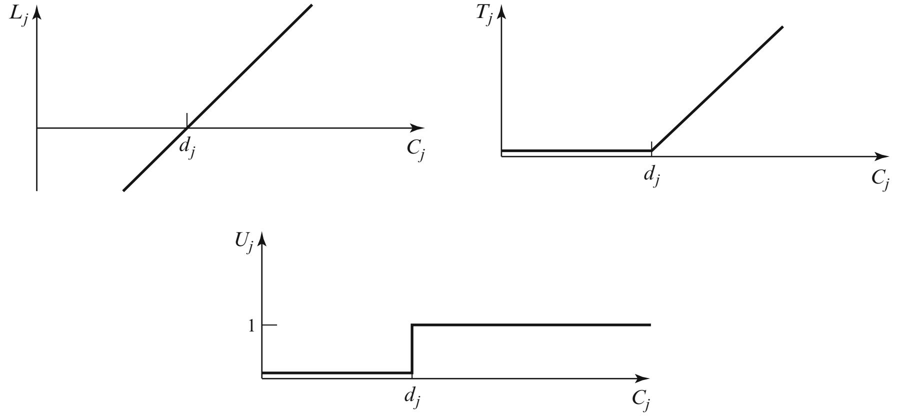

Fig. 2.1 Due date related penalty functions

The objective to be minimized is always a function of the completion times of the jobs, which, of course, depend on the schedule. The completion time of the operation of job j on machine i is denoted by C_{ij}. The time job j exits the system (that is, its completion time on the last machine on which it requires processing) is denoted by C_j. The objective may also be a function of the due dates. The *lateness* of job j is defined as

$$L_j = C_j - d_j,$$

which is positive when job j is completed late and negative when it is completed early. The *tardiness* of job j is defined as

$$T_j = \max(C_j - d_j, 0) = \max(L_j, 0).$$

The difference between the tardiness and the lateness lies in the fact that the tardiness never is negative. The *unit penalty* of job j is defined as

$$U_j = \begin{cases} 1 \text{ if } C_j > d_j \\ 0 \text{ otherwise} \end{cases}$$

The lateness, the tardiness, and the unit penalty are the three basic due date related penalty functions considered in this book. The shape of these functions are depicted in Figure 2.1.

Examples of possible objective functions to be minimized are:

Makespan ($C_{\max}$) The makespan, defined as $\max(C_1, \ldots, C_n)$, is equivalent to the completion time of the last job to leave the system. A minimum makespan usually implies a good utilization of the machine(s).

Maximum Lateness ($L_{\max}$) The maximum lateness, $L_{\max}$, is defined as $\max(L_1, \ldots, L_n)$. It measures the worst violation of the due dates.

Total weighted completion time ($\sum w_j C_j$) The sum of the weighted completion times of the n jobs gives an indication of the total holding or inventory costs incurred by the schedule. The sum of the completion times is in the literature often referred to as the flow time. The total weighted completion time is then referred to as the weighted flow time.

Discounted total weighted completion time ($\sum w_j(1 - e^{-rC_j})$) This is a more general cost function than the previous one, where costs are discounted at a rate of r, $0 < r < 1$, per unit time. That is, if job j is not completed by time t an additional cost $w_j r e^{-rt} dt$ is incurred over the period $[t, t + dt]$. If job j is completed at time t, the total cost incurred over the period $[\,0, t\,]$ is $w_j(1 - e^{-rt})$. The value of r is usually close to 0, say 0.1 or 10 %.

Total weighted tardiness ($\sum w_j T_j$) This is also a more general cost function than the total weighted completion time.

Weighted number of tardy jobs ($\sum w_j U_j$) The weighted number of tardy jobs is not only a measure of academic interest, it is often an objective in practice as it is a measure that can be recorded very easily.

All the objective functions above are so-called *regular* performance measures. A regular performance measure is a function that is *non-decreasing* in $C_1, \ldots, C_n$. Recently researchers have begun to study objective functions that are not regular. For example, when job j has a due date d_j, it may be subject to an earliness penalty, where the *earliness* of job j is defined as

$$E_j = \max(d_j - C_j, 0).$$

This earliness penalty is *non-increasing* in C_j. An objective such as the total earliness plus the total tardiness, i.e.,

$$\sum_{j=1}^{n} E_j + \sum_{j=1}^{n} T_j,$$

is therefore not regular. A more general objective that is not regular is the total weighted earliness plus the total weighted tardiness, i.e.,

$$\sum_{j=1}^{n} w_j' E_j + \sum_{j=1}^{n} w_j'' T_j.$$

The weight associated with the earliness of job j (w_j') may be different from the weight associated with the tardiness of job j (w_j'').

2.2 Examples

The following examples illustrate the notation:

Example 2.2.1 (A Flexible Flow Shop). $FFc \mid r_j \mid \sum w_j T_j$ denotes a flexible flow shop. The jobs have release dates and due dates and the objective is the minimization of the total weighted tardiness. Example 1.1.1 in Section 1.1 (the paper bag factory) can be modeled as such. Actually, the problem described in Section 1.1 has some additional characteristics including sequence dependent setup times at each of the three stages. In addition, the processing time of job j on machine i has a special structure: it depends on the number of bags and on the speed of the machine. ||

Example 2.2.2 (A Flexible Job Shop). $FJc \mid r_j, s_{ijk}, rcrc \mid \sum w_j T_j$ refers to a flexible job shop with c work centers. The jobs have different release dates and are subject to sequence dependent setup times that are machine dependent. There is recirculation, so a job may visit a work center more than once. The objective is to minimize the total weighted tardiness. It is clear that this problem is a more general problem than the one described in the previous example. Example 1.1.2 in Section 1.1 (the semiconductor manufacturing facility) can be modeled as such. ||

Example 2.2.3 (A Parallel Machine Environment). $Pm \mid r_j, M_j \mid \sum w_j T_j$ denotes a system with m machines in parallel. Job j arrives at release date r_j and has to leave by the due date d_j. Job j may be processed only on one of the machines belonging to the subset M_j. If job j is not completed in time a penalty $w_j T_j$ is incurred. This model can be used for the gate assignment problem described in Example 1.1.3. ||

Example 2.2.4 (A Single Machine Environment). $1 \mid r_j, prmp \mid \sum w_j C_j$ denotes a single machine system with job j entering the system at its release date r_j. Preemptions are allowed. The objective to be minimized is the sum of the weighted completion times. This model can be used to study the deterministic counterpart of the problem described in Example 1.1.4. ||

Example 2.2.5 (Sequence Dependent Setup Times). $1 \mid s_{jk} \mid C_{\max}$ denotes a single machine system with n jobs subject to sequence dependent setup times, where the objective is to minimize the makespan. It is well known that this problem is equivalent to the so-called *Travelling Salesman Problem (TSP)*, where a salesman has to tour n cities in such a way that the total distance traveled is minimized (see Appendix D for a formal definition of the TSP). ||

Example 2.2.6 (A Project). $P\infty \mid prec \mid C_{\max}$ denotes a scheduling problem with n jobs subject to precedence constraints and an unlimited number of machines (or resources) in parallel. The total time of the entire project has to be minimized. This type of problem is very common in project planning in the construction industry and has led to techniques such as the *Critical* Path Method (CPM) and the *Project Evaluation and Review Technique (PERT).* ||

Example 2.2.7 (A Flow Shop). $Fm \mid p_{ij} = p_j \mid \sum w_j C_j$ denotes a *proportionate* flow shop environment with m machines in series; the processing times of job j on all m machines are identical and equal to p_j (hence the term proportionate). The objective is to find the order in which the n jobs go through the system so that the sum of the weighted completion times is minimized. ||

Example 2.2.8 (A Job Shop). $Jm \mid\mid C_{\max}$ denotes a job shop problem with m machines. There is no recirculation, so a job visits each machine at most once. The objective is to minimize the makespan. This problem is considered a classic in the scheduling literature and has received an enormous amount of attention. ||

Of course, there are many scheduling models that are not captured by this framework. One can define, for example, a more general flexible job shop in which each work center consists of a number of unrelated machines in parallel. When a job on its route through the system arrives at a bank of unrelated machines, it may be processed on any one of the machines, but its processing time now depends on the machine on which it is processed.

One can also define a model that is a mixture of a job shop and an open shop. The routes of some jobs are fixed, while the routes of other jobs are (partially) open.

The framework described in Section 2.1 has been designed primarily for models with a single objective. Most research in the past has concentrated on models with a single objective. Recently, researchers have begun studying models with multiple objectives as well.

Various other scheduling features, that are not mentioned here, have been studied and analyzed in the literature. Such features include periodic or cyclic scheduling, personnel scheduling, and resource constrained scheduling.

2.3 Classes of Schedules

In scheduling terminology a distinction is often made between a *sequence*, a *schedule* and a *scheduling policy.* A sequence usually corresponds to a permutation of the n jobs or the order in which jobs are to be processed on a given machine. A schedule usually refers to an allocation of jobs within a more complicated setting of machines, allowing possibly for preemptions of jobs by other jobs that are released at later points in time. The concept of a scheduling policy is often used in stochastic settings: a policy prescribes an appropriate action for any one of the states the system may be in. In deterministic models usually only sequences or schedules are of importance.

Assumptions have to be made with regard to what the scheduler may and may not do when (s)he generates a schedule. For example, it may be the case that a schedule may not have any *unforced idleness* on any machine. This class of schedules can be defined as follows.

Definition 2.3.1 (Non-delay Schedule). A feasible schedule is called non-delay if no machine is kept idle while an operation is waiting for processing.

Requiring a schedule to be non-delay is equivalent to prohibiting *unforced idleness.* For many models, including those that allow preemptions and have regular objective functions, there are optimal schedules that are non-delay. For many models considered in this part of the book the goal is to find an optimal schedule that is non-delay. However, there *are* models where it may be advantageous to have periods of unforced idleness.

A smaller class of schedules, within the class of all non-delay schedules, is the class of non-preemptive non-delay schedules. Non-preemptive non-delay schedules may lead to some interesting and unexpected anomalies.

Example 2.3.2 (A Scheduling Anomaly). Consider an instance of $P2 \mid prec \mid C_{\max}$ with 10 jobs and the following processing times.

jobs	1	2	3	4	5	6	7	8	9	10
p_j	8	7	7	2	3	2	2	8	8	15

The jobs are subject to the precedence constraints depicted in Figure 2.2. The makespan of the non-delay schedule depicted in Figure 2.3.a is 31 and the schedule is clearly optimal.

One would expect that, if each one of the ten processing times is reduced by one time unit, the makespan would be less than 31. However, requiring the schedule to be non-delay results in the schedule depicted in Figure 2.3.b with a makespan of 32.

Suppose that an additional machine is made available and that there are now three machines instead of two. One would again expect the makespan with the original set of processing times to be less than 31. Again, the non-delay requirement has an unexpected effect: the makespan is now 36. ||

Some heuristic procedures and algorithms for job shops are based on the construction of non-preemptive schedules with certain special properties. Two classes of non-preemptive schedules are of importance for certain algorithmic procedures for job shops.

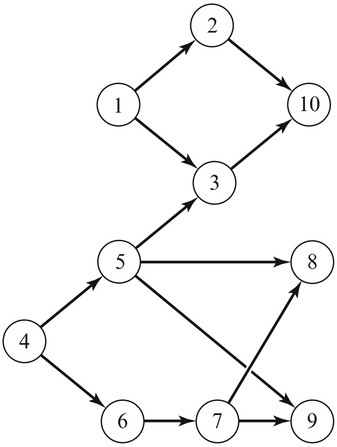

Fig. 2.2 Precedence constraints graph for Example 2.3.2.

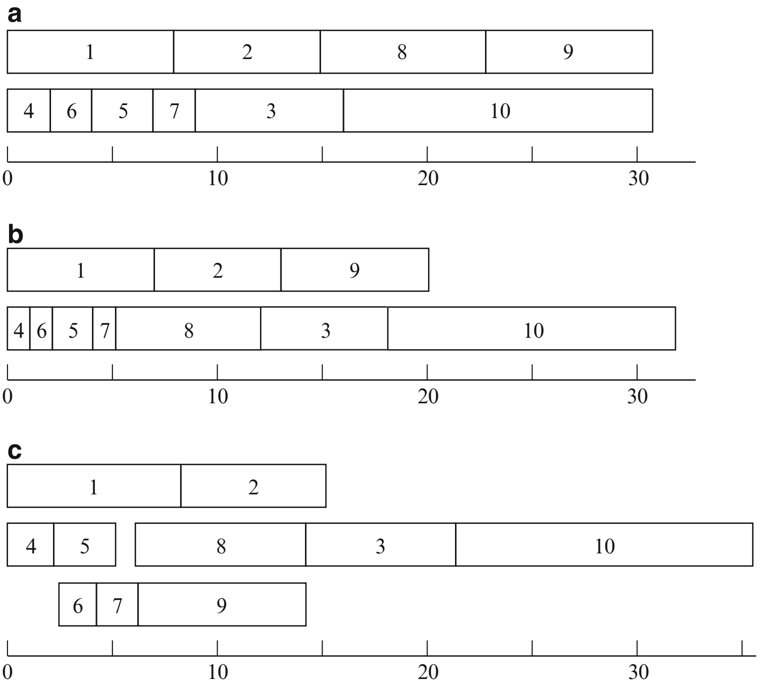

Fig. 2.3 Gantt charts of non-delay schedules: (a) Original schedule (b) Processing times one unit shorter (c) Original processing times and three machines

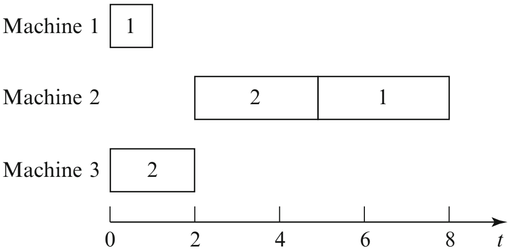

Fig. 2.4 An active schedule that is not non-delay.

Definition 2.3.3 (Active Schedule). A feasible non-preemptive schedule is called active if it is not possible to construct another schedule, through changes in the order of processing on the machines, with at least one operation finishing earlier and no operation finishing later.

In other words, a schedule is active if no operation can be put into an empty hole earlier in the schedule while preserving feasibility. A non-preemptive non-delay schedule has to be active but the reverse is not necessarily true. The following example describes a schedule that is active but not non-delay.

Example 2.3.4 (An Active Schedule). Consider a job shop with three machines and two jobs. Job 1 needs one time unit on machine 1 and 3 time units on machine 2. Job 2 needs 2 time units on machine 3 and 3 time units on machine 2. Both jobs have to be processed last on machine 2. Consider the schedule which processes job 2 on machine 2 before job 1 (see Figure 2.4). It is clear that this schedule is active; reversing the sequence of the two jobs on machine 2 postpones the processing of job 2. However, the schedule is *not* non-delay. Machine 2 remains idle till time 2, while there is already a job available for processing at time 1. ||

It can be shown that, when the objective γ is regular, there exists for $Jm \mid\mid \gamma$ an optimal schedule that is active.

An even larger class of non-preemptive schedules can be defined as follows.

Definition 2.3.5 (Semi-Active Schedule). A feasible non-preemptive schedule is called semi-active if no operation can be completed earlier without changing the order of processing on any one of the machines.

It is clear that an active schedule has to be semi-active. However, the reverse is not necessarily true.

Example 2.3.6 (A Semi-Active Schedule). Consider again a job shop with three machines and two jobs. The routing of the two jobs is the same as in the previous example. The processing times of job 1 on machines 1 and 2 are both equal to 1. The processing times of job 2 on machines 2 and 3 are both equal to 2.

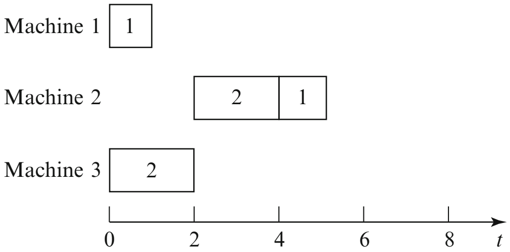

Fig. 2.5 A semi-active schedule that is not active.

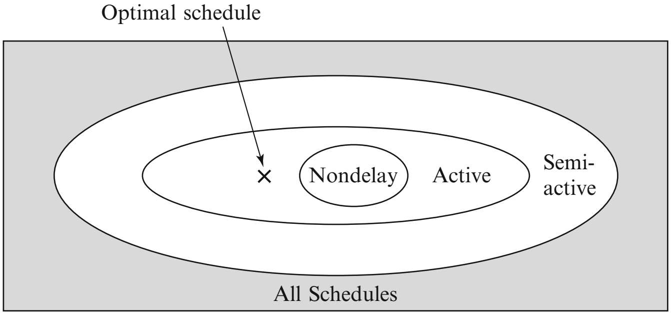

Fig. 2.6 Venn diagram of classes of non-preemptive schedules for job shops

Consider the schedule under which job 2 is processed on machine 2 before job 1 (see Figure 2.5). This implies that job 2 starts its processing on machine 2 at time 2 and job 1 starts its processing on machine 2 at time 4. This schedule is semi-active. However, it is not active, as job 1 can be processed on machine 2 without delaying the processing of job 2 on machine 2.

An example of a schedule that is not even semi-active can be constructed easily. Postpone the start of the processing of job 1 on machine 2 for one time unit, i.e., machine 2 is kept idle for one unit of time between the processing of jobs 2 and 1. Clearly, this schedule is not even semi-active. ||

Figure 2.6 shows a Venn diagram of the three classes of non-preemptive schedules: the non-preemptive non-delay schedules, the active schedules, and the semi-active schedules.

2.4 Complexity Hierarchy

Often, an algorithm for one scheduling problem can be applied to another scheduling problem as well. For example, $1 \mid\mid \sum C_j$ is a special case of $1 \mid\mid \sum w_j C_j$ and a procedure for $1 \mid\mid \sum w_j C_j$ can, of course, also be used for $1 \mid\mid \sum C_j$. In complexity terminology it is then said that $1 \mid\mid \sum C_j$ *reduces* to $1 \mid\mid \sum w_j C_j$. This is usually denoted by

$$1 \mid\mid \sum C_j \;\; \propto \;\; 1 \mid\mid \sum w_j C_j.$$

Based on this concept a chain of reductions can be established. For example,

$$1 \mid\mid \sum C_j \;\; \propto \;\; 1 \mid\mid \sum w_j C_j \;\; \propto \;\; Pm \mid\mid \sum w_j C_j \;\; \propto \;\; Qm \mid prec \mid \sum w_j C_j.$$

Of course, there are also many problems that are not comparable with one another. For example, $Pm \mid\mid \sum w_j T_j$ is not comparable to $Jm \mid\mid C_{\max}$.

A considerable effort has been made to establish a problem hierarchy describing the relationships between the hundreds of scheduling problems. In the comparisons between the complexities of the different scheduling problems it is of interest to know how a change in a single element in the classification of a problem affects its complexity. In Figure 2.7 a number of graphs are exhibited that help determine the complexity hierarchy of deterministic scheduling problems. Most of the hierarchy depicted in these graphs is relatively straightforward. However, two of the relationships may need some explaining, namely

$$\alpha \mid \beta \mid L_{\max} \;\; \propto \;\; \alpha \mid \beta \mid \sum U_j$$

and

$$\alpha \mid \beta \mid L_{\max} \;\; \propto \;\; \alpha \mid \beta \mid \sum T_j.$$

It can, indeed, be shown that a procedure for $\alpha \mid \beta \mid \sum U_j$ and a procedure for $\alpha \mid \beta \mid \sum T_j$ can be applied to $\alpha \mid \beta \mid L_{\max}$ with only minor modifications (see Exercise 2.23).

A significant amount of research in deterministic scheduling has been devoted to finding efficient, so-called polynomial time, algorithms for scheduling problems. However, many scheduling problems do not have a polynomial time algorithm; these problems are the so-called *NP-hard* problems. Verifying that a problem is NP-hard requires a formal mathematical proof (see Appendix D).

Research in the past has focused in particular on the borderline between polynomial time solvable problems and NP-hard problems. For example, in the string of problems described above, $1 \mid\mid \sum w_j C_j$ can be solved in polynomial time, whereas $Pm \mid\mid \sum w_j C_j$ is NP-hard, which implies that $Qm \mid prec \mid \sum w_j C_j$ is also NP-hard. The following examples illustrate the borderlines between easy and hard problems within given sets of problems.

Example 2.4.1 (A Complexity Hierarchy). Consider the problems

(i) $1 \mid\mid C_{\max}$,

(ii) $P2 \mid\mid C_{\max}$,

(iii) $F2 \mid\mid C_{\max}$,

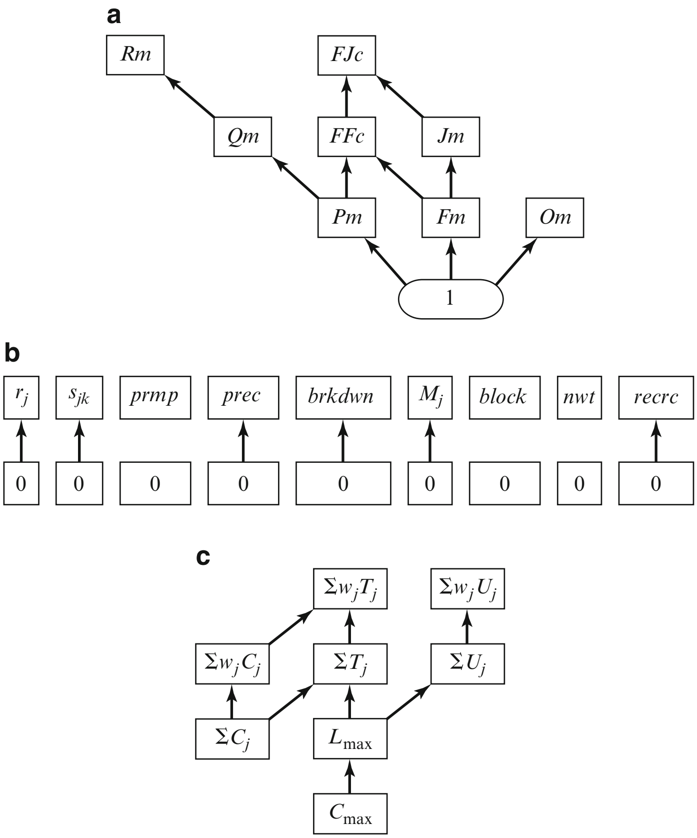

Fig. 2.7 Complexity hierarchies of deterministic scheduling problems: (a) Machine environments (b) Processing restrictions and constraints (c) Objective functions

(iv) $Jm \mid\mid C_{\max}$,
(v) $FFc \mid\mid C_{\max}$.

The complexity hierarchy is depicted in Figure 2.8. $||$

Example 2.4.2 (A Complexity Hierarchy). Consider the problems

(i) $1 \mid\mid L_{\max}$,
(ii) $1 \mid prmp \mid L_{\max}$,
(iii) $1 \mid r_j \mid L_{\max}$,
(iv) $1 \mid r_j, prmp \mid L_{\max}$,
(v) $Pm \mid\mid L_{\max}$.

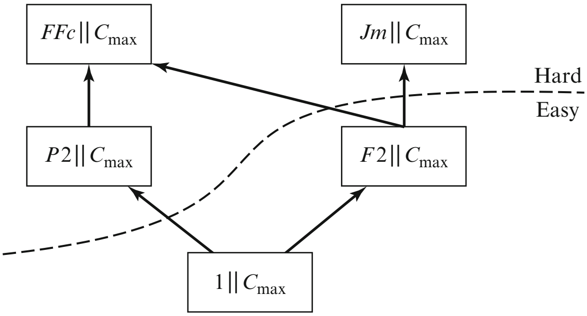

Fig. 2.8 Complexity hierarchy of problems in Example 2.4.1

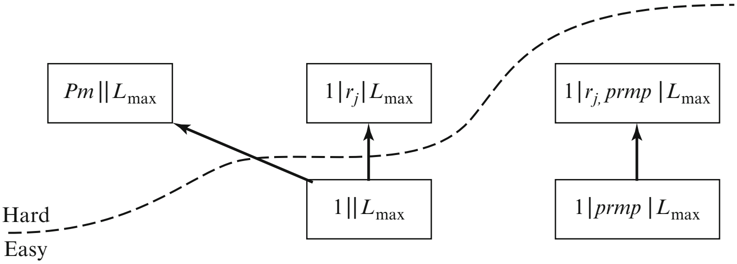

Fig. 2.9 Complexity hierarchy of problems in Example 2.4.2

The complexity hierarchy is depicted in Figure 2.9. ||

Exercises (Computational)

2.1. Consider the instance of $1 \| \sum w_j C_j$ with the following processing times and weights.

jobs	1	2	3	4
w_j	6	11	9	5
p_j	3	5	7	4

(a) Find the optimal sequence and compute the value of the objective.

(b) Give an argument for positioning jobs with larger weight more towards the beginning of the sequence and jobs with smaller weight more towards the end of the sequence.

(c) Give an argument for positioning jobs with smaller processing time more towards the beginning of the sequence and jobs with larger processing time more towards the end of the sequence.

(d) Determine which one of the following two generic rules is the most suitable for the problem:

(i) sequence the jobs in decreasing order of $w_j - p_j$;

(ii) sequence the jobs in decreasing order of w_j/p_j.

2.2. Consider the instance of $1 \mid\mid L_{\max}$ with the following processing times and due dates.

jobs	1	2	3	4
p_j	5	4	3	6
d_j	3	5	11	12

(a) Find the optimal sequence and compute the value of the objective.

(b) Give an argument for positioning jobs with earlier due dates more towards the beginning of the sequence and jobs with later due dates more towards the end of the sequence.

(c) Give an argument for positioning jobs with smaller processing time more towards the beginning of the sequence and jobs with larger processing time more towards the end of the sequence.

(d) Determine which one of the following four rules is the most suitable generic rule for the problem:

(i) sequence the jobs in increasing order of $d_j + p_j$;

(ii) sequence the jobs in increasing order of $d_j p_j$;

(iii) sequence the jobs in increasing order of d_j;

(iv) sequence the jobs in increasing order of p_j.

2.3. Consider the instance of $1 \mid\mid \sum U_j$ with the following processing times and due dates.

jobs	1	2	3	4
p_j	7	6	4	8
d_j	8	9	11	14

(a) Find all optimal sequences and compute the value of the objective.

(b) Formulate a generic rule based on the due dates and processing times that yields an optimal sequence for any instance.

2.4. Consider the instance of $1 \mid\mid \sum T_j$ with the following processing times and due dates.

jobs	1	2	3	4
p_j	7	6	8	4
d_j	8	9	10	14

(a) Find all optimal sequences.

(b) Formulate a generic rule that is a function of the due dates and processing times that yields an optimal sequence for any instance.

2.5. Find the optimal sequence for $P5 \mid\mid C_{\max}$ with the following 11 jobs.

jobs	1	2	3	4	5	6	7	8	9	10	11
p_j	9	9	8	8	7	7	6	6	5	5	5

2.6. Consider the instance of $F2 \mid prmu \mid C_{\max}$ with the following processing times.

jobs	1	2	3	4
p_{1j}	8	6	4	12
p_{2j}	4	9	10	6

Find all optimal sequences and determine the makespan under an optimal sequence.

2.7. Consider the instance of $F2 \mid block \mid C_{\max}$ with the same jobs and the same processing times as in Exercise 2.6. There is no (zero) buffer between the two machines. Find all optimal sequences and compute the makespan under an optimal sequence.

2.8. Consider the instance of $F2 \mid nwt \mid C_{\max}$ with the same jobs and the same processing times as in Exercise 2.6. Find all optimal sequences and compute the makespan under an optimal sequence.

2.9. Consider the instance of $O2 \mid\mid C_{\max}$ with 4 jobs. The processing times of the four jobs on the two machines are again as in Exercise 2.6. Find all optimal schedules and compute the makespan under an optimal schedule.

2.10. Consider the instance of $J2 \mid\mid C_{\max}$ with 4 jobs. The processing times of the four jobs on the two machines are again as in Exercise 2.6. Jobs 1 and 2 have to be processed first on machine 1 and then on machine 2, while jobs 3 and 4 have to be processed first on machine 2 and then on machine 1. Find all optimal schedules and determine the makespan under an optimal schedule.

Exercises (Theory)

2.11. Explain why $\alpha \mid p_j = 1, r_j \mid \gamma$ is easier than $\alpha \mid prmp, r_j \mid \gamma$ when all processing times, release dates and due dates are integer.

2.12. Consider $1 \mid s_{jk} = a_k + b_j \mid C_{\max}$. That is, job j has two parameters associated with it, namely a_j and b_j. If job j is followed by job k, there is a setup time $s_{jk} = a_k + b_j$ required before the start of job k's processing. The setup time of the first job in the sequence, s_{0k} is a_k, while the "clean-up" time at the completion of the last job in the sequence, s_{j0}, is b_j. Show that this problem is equivalent to $1 \mid\mid C_{\max}$ and that the makespan therefore does not depend on the sequence. Find an expression for the makespan.

2.13. Show that $1 \mid s_{jk} \mid C_{\max}$ is equivalent to the following Travelling Salesman Problem: A travelling salesman starts out from city 0, visits cities $1, 2, \ldots, n$, and returns to city 0, while minimizing the total distance travelled. The distance from city 0 to city k is s_{0k}; the distance from city j to city k is s_{jk} and the distance from city j to city 0 is s_{j0}.

2.14. Show that $1 \mid brkdwn, prmp \mid \sum w_j C_j$ reduces to $1 \mid r_j, prmp \mid \sum w_j C_j$.

2.15. Show that $1 \mid p_j = 1 \mid \sum w_j T_j$ and $1 \mid p_j = 1 \mid L_{\max}$ are equivalent to the *assignment* problem (see Appendix A for a definition of the assignment problem).

2.16. Show that $Pm \mid p_j = 1 \mid \sum w_j T_j$ and $Pm \mid p_j = 1 \mid L_{\max}$ are equivalent to the *transportation* problem (see Appendix A for a definition of the transportation problem).

2.17. Consider $P \mid\mid C_{\max}$. Show that for any non-delay schedule the following inequalities hold:

$$\frac{\sum p_j}{m} \leq C_{\max} \leq 2 \times \max\left(p_1, \ldots, p_n, \frac{\sum p_j}{m}\right).$$

2.18. Show how $Pm \mid M_j \mid \gamma$ reduces to $Rm \mid\mid \gamma$.

2.19. Show that $F2 \mid block \mid C_{\max}$ is equivalent to $F2 \mid nwt \mid C_{\max}$ and show that both problems are special cases of $1 \mid s_{jk} \mid C_{\max}$ and therefore special cases of the Travelling Salesman Problem.

2.20. Consider an instance of $Om \mid \beta \mid \gamma$ and an instance of $Fm \mid \beta \mid \gamma$. The two instances have the same number of machines, the same number of jobs, and the jobs have the same processing times on the m machines. The two instances are completely identical with the exception that one instance is an open shop and the other instance a flow shop. Show that the value of the objective under the optimal sequence in the flow shop is at least as large as the value of the objective under the optimal sequence in the open shop.

2.21. Consider $O2 \mid\mid C_{\max}$. Show that

$$C_{\max} \geq \max \left(\sum_{j=1}^{n} p_{1j}, \sum_{j=1}^{n} p_{2j} \right).$$

Find an instance of this problem where the optimal makespan is *strictly* larger than the RHS.

2.22. Describe the complexity relationships between the problems

(i) $1 \mid\mid \sum w_j C_j$,
(ii) $1 \mid d_j = d \mid \sum w_j T_j$,
(iii) $1 \mid p_j = 1 \mid \sum w_j T_j$,
(iv) $1 \mid\mid \sum w_j T_j$,
(v) $Pm \mid p_j = 1 \mid \sum w_j T_j$,
(vi) $Pm \mid\mid \sum w_j T_j$.

2.23. Show that $\alpha \mid \beta \mid L_{\max}$ reduces to $\alpha \mid \beta \mid \sum T_j$ as well as to $\alpha \mid \beta \mid \sum U_j$. (*Hint:* Note that if the minimum $L_{\max}$ is zero, the optimal solution with regard to $\sum U_j$ and $\sum T_j$ is zero as well. It suffices to show that a polynomial time procedure for $\alpha \mid \beta \mid \sum U_j$ can be adapted easily for application to $\alpha \mid \beta \mid L_{\max}$. This can be done through a parametric analysis on the d_j, i.e., solve $\alpha \mid \beta \mid \sum U_j$ with due dates $d_j + z$ and vary z.)

Comments and References

One of the first classification schemes for scheduling problems appeared in Conway, Maxwell, and Miller (1967). Lawler, Lenstra, and Rinnooy Kan (1982), in their survey paper, modified and refined this scheme extensively. Herrmann, Lee, and Snowdon (1993) made another round of extensions. The framework presented here is another variation of the Lawler, Lenstra, and Rinnooy Kan (1982) notation, with a slightly different emphasis.

For a survey of scheduling problems subject to availability constraints (*brkdwn*), see Lee (2004). For surveys on scheduling problems with non-regular objective functions, see Raghavachari (1988) and Baker and Scudder (1990). For a survey of scheduling problems with job families and scheduling problems with batch processing, see Potts and Kovalyov (2000).

The definitions of non-delay, active, and semi-active schedules have been around for a long time; see, for example, Giffler and Thompson (1960) and French (1982) for a comprehensive overview of classes of schedules. Example 2.3.2, which illustrates some of the anomalies of non-delay schedules, is due to Graham (1966).

The complexity hierarchy of scheduling problems is motivated primarily by the work of Rinnooy Kan (1976), Lenstra (1977), Lageweg, Lawler, Lenstra, and Rinnooy Kan (1981, 1982) and Lawler, Lenstra, Rinnooy Kan, and Shmoys (1993). For more on reducibility in scheduling, see Timkovsky (2004).