1 Introduction

Scheduling is a decision-making process that is used on a regular basis in manufacturing and service industries. For example, scheduling plays an important role in the production of micro-electronics manufacturing, in operator shift scheduling in call centers, and in the coordination of health care activities in hospitals. Scheduling processes in manufacturing industries tend to be different from scheduling processes in service industries. Scheduling processes in manufacturing industries are often more structured and better understood than those in service industries. The mathematical models that have led to the many scheduling algorithms that are applicable in manufacturing often fit very well within an established and well-organized framework. Scheduling in manufacturing industries may at times be referred to as machine scheduling, assembly line sequencing, or resource-constrained project scheduling. Scheduling processes in service industries vary in nature and may be more industry dependent. The service industries in which scheduling plays an important role include the transportation industries, the health care industry, the hospitality industries, and the entertainment industry. A scheduling process in any given service industry typically has its own peculiarities and idiosyncrasies.

Since it would be impossible to present an overview and a description of all the different types of scheduling processes that can take place in every possible manufacturing or services setting in a single chapter, in this chapter, we only consider four basic scheduling paradigms in detail. Two of these paradigms are prevalent in a variety of different manufacturing industries, and the other two, in various different service industries. In the conclusion, we refer to some other types of scheduling environments.

With regard to scheduling processes in manufacturing, we consider two important paradigms, namely,

(i) job shop scheduling and
(ii) resource constrained project scheduling.

Job shop scheduling plays an important role in a plethora of production environments (e.g., semiconductor manufacturing, printed circuit board manufacturing, and so on). Resource constrained project scheduling plays an important role in the manufacturing and construction of, for example, ocean liners and power plants.
With regard to scheduling processes in the service industries, we elaborate in what follows on two other paradigms, namely,

(i) personnel (shift) scheduling and
(ii) appointment scheduling.

Because of the typical and very variable demand for services in many different industries, shift scheduling in the service industries is significantly more challenging than shift scheduling in manufacturing. That is the main reason why research in personnel scheduling most often focuses on the service industries. Shift scheduling plays an important role in operator scheduling in call centers as well as in health care (e.g., nurse scheduling in hospitals). Appointment scheduling plays a crucial role in the scheduling of meetings, doctor appointments, surgeries, and so on.

There is an extensive amount of literature on scheduling theory and applications. In the last couple of decades, many books and survey papers appeared, covering all aspects of scheduling theory and applications. The more generally oriented books on scheduling include Baker and Trietsch (2009), Baptiste et al. (2001), Blazewicz et al. (2001), Brucker (2006), Brucker and Knust (2012), Pinedo (2009), and Pinedo (2016). The survey papers include those by Graves (1981) and Smith (1992).

1.1 Classifications of Scheduling Problems

In the literature, scheduling problems have been classified in various different ways. One way of classifying scheduling problems is based on when the exact data concerning a scheduling instance (i.e., release dates, processing times, etc.) become available to the scheduler. According to this criterion, there are two main classes of scheduling problems, namely, static problems and dynamic problems. Static scheduling assumes that all the information regarding the entire process (i.e., information with regard to the jobs as well as with regard to the machines) is already available at time zero and will not change over time. Therefore, the entire schedule can already be determined at time zero when the process starts. Dynamic scheduling assumes that new information may become available while the process evolves. For example, the exact processing time of a job may not be known beforehand, but it becomes known upon its completion. In a dynamic scheduling environment, a schedule is therefore being created while the process evolves. In the literature, static and dynamic scheduling problems are often also referred to as offline and online scheduling problems, respectively. Examples of static scheduling processes can often be found in production scheduling in manufacturing industries. The jobs to be processed within a scheduling horizon are selected in advance and all information concerning these jobs (e.g., their processing times) as well as all information concerning the machines (e.g., availability times) are known beforehand. Dynamic scheduling may occur, for example, in call centers. In such an environment, customers’ calls are regarded as jobs and the information regarding a job only becomes known at the moment that the job comes in.

Another method of classifying scheduling problems is based on the probability distributions of quantities of interest, such as processing times, release dates, due dates, and so on. One can make a distinction between deterministic and stochastic scheduling. In deterministic scheduling, job characteristics such as release dates, processing times, and due dates are all known with certainty in advance. In stochastic scheduling, job characteristics such as processing times are known to be random variables from given probability distributions with known parameters. The actual processing time of a job only becomes known upon its completion. In deterministic scheduling, the value of an objective function can be computed precisely beforehand. In stochastic
scheduling, one may only be able to compute the objective function in expectation. However, such an expectation is typically hard to compute. Scheduling problems in manufacturing tend to be more static and deterministic whereas scheduling problems in services tend to be more dynamic and stochastic. However, there are also settings that give rise to deterministic problems that are dynamic and settings that give rise to stochastic problems that are static.

This chapter focuses mainly on static (offline) deterministic scheduling problems. This class of problems has received most attention in the literature and the insights obtained through this research have often been useful in the analysis of related problems in the other classes of scheduling problems.

This chapter is organized as follows: In Section 2, we provide an overview of mathematical and computational preliminaries and the fundamental algorithmic frameworks that have wide applicability. In Section 3, we focus on selected topics in manufacturing scheduling, and in Section 4, we focus on selected topics in services scheduling. In Section 5, we discuss implementation issues, and in the last section, we present our conclusions and directions for future research.

2 Preliminaries and Fundamentals

2.1 Computational Complexity of Scheduling Problems

A schedule is a solution of a scheduling problem. It specifies when (at what time) and where (on which machine) each job is processed. In many scheduling problems, the objective function is to minimize a function of the completion times of the jobs. Two types of objective functions are common: (i) the sum of job values that are a function of their completion times (often referred to as a “min-sum” type of objective) and (ii) the maximum of job values that are a function of their completion times (often referred to as a “min-max” type of objective).

Static deterministic scheduling problems can be classified according to their computational complexity. Some scheduling problems tend to be relatively easy and can be solved without too much difficulty. In mathematical terms it is said that these easy problems can be solved in polynomial time because the running time to solve the problem is a polynomial function of the size of an instance of the problem (e.g., the number of jobs involved). Such problems are often referred to as tractable. That is to say that they can be solved fast in generating optimal schedules, using only a limited amount of computer time even when a large number of activities or jobs are involved.

Some of such tractable scheduling problems can be solved by applying a simple dispatching or priority rule. A classic example of a simple priority rule in a scheduling environment is the so-called Weighted Shortest Processing Time (WSPT) rule. Consider a single machine and n jobs. Job j has a processing time pj and a weight (priority level) wj, and its completion time in a schedule is denoted by Cj, j = 1, 2, . . . , n. If the n jobs have to be scheduled on the machine in such a way that the total weighted completion time (Σj wj Cj) has to be minimized, then the jobs have to be sequenced according to the WSPT rule (i.e., in decreasing order of wj / pj). The total weighted completion time is an example of an objective of the min-sum type. There are other and more complicated scheduling problems that are still tractable and also can be solved in polynomial time using algorithms that are a little bit more elaborate than simple priority rules. The techniques used for solving such “easy” scheduling problems may often be based on dynamic programming or linear programming techniques.

However, most static deterministic scheduling problems tend to be quite hard in practice and may not be solvable in polynomial time. Such problems are often referred to as intractable and are in mathematical terms called NP-Hard. (The concept of NP-Hardness is based on a formal
mathematical definition that is beyond the scope of this chapter. However, it is generally believed that an algorithm for solving an NP-Hard problem cannot run in polynomial time.) NP-Hard problems can only be solved via elaborate and time-consuming optimization techniques.

Before describing the various solution approaches, it will be helpful to discuss the relationships between different problems as far as their complexities and solution methodologies are concerned.

Consider two scheduling problems. Let problem $P_1$ be a scheduling problem to minimize the total completion time on a single machine and let problem $P_2$ be a scheduling problem to minimize the total weighted completion time on a single machine. Thus, problem $P_2$ becomes problem $P_1$ when the weights of all jobs are made equal to one another. In such a case, we can say that problem $P_1$ is a special case of problem $P_2$ and problem $P_2$ is more general than problem $P_1$. In other words, problem $P_2$ is at least as hard as problem $P_1$. Specifically, if a special case is hard, then a more general case is hard too. Similarly, if a more general case is easy, then the special case will be easy as well. Such relationships are useful in determining the complexities of scheduling problems. As for solution methodology, a solution methodology for a general case is always applicable to a special case as well. On the other hand, a solution methodology for a special case may not be directly applicable to a more general case. It may need to be generalized; if it can be applied, the quality of the solution may be lower, or it may not be applicable at all, and an entirely different approach may have to be developed.

### 2.2 Solution Methodologies

Now, consider the solution methodologies. There are two criteria for evaluating the effectiveness of a solution methodology: the solution quality and the solution time. It is desirable to have a solution methodology that produces an optimal (or near-optimal) schedule in a very short amount of time. First of all, for tractable problems, we already know that they can be solved optimally in polynomial time. Thus, the research focuses on reducing the computation time mathematically and practically.

Intractable problems can only be solved via more elaborate optimization techniques. In the literature, the various techniques are typically categorized as (i) exact algorithms, (ii) approximation algorithms, and (iii) heuristic algorithms.

(i) Exact algorithms include Dynamic Programming as well as Branch-and-Bound techniques, which may be used for various forms of mathematical programming formulations, (e.g., Mixed Integer Programming (MIP) formulations, which may use various types of decision variables such as time-indexed variables or sequencing variables). Most exact algorithmic approaches are based on an efficient enumeration of the solution space while trying to avoid doing any unnecessary searches in the solution space. Only small instances can be solved optimally by exact algorithms within a reasonable time frame.

(ii) Approximation algorithms are often based on priority rules because of their ease of implementation and analysis. Sometimes, the problem is reduced to an easier problem that is similar in order to get an optimal solution for the modified problem and the optimal solution to the modified problem is then converted to a feasible solution to the original problem. Worst-case analysis is used mainly for evaluating the performance of approximation algorithms. It may be possible to show that the ratio of the objective function value of the solution obtained via an approximation algorithm to the optimal objective function value is no more than a certain value even though the optimal objective function value is not known. The minimum possible value of such a ratio is referred to as a worst-case performance ratio when offline scheduling problems are considered and as a competitive ratio when online scheduling problems are considered. We consider in the
next section the application of the WSPT rule in an environment with \( m \) identical machines in parallel as an example of an approximation algorithm.

(iii) Heuristics have to be designed when problems are too complicated to allow for either exact algorithms or approximation algorithms. A heuristic may incorporate rules of thumb that are based on certain characteristics of the problem. It may also decompose the problem into sub-problems and then solve these subproblems sequentially in order to gain computational efficiency with some sacrifice in the quality of the solution. Since in many cases it is hard to evaluate the solution quality of a heuristic in a theoretical manner, the performance of a heuristic is often evaluated through experimentation with randomly generated data sets and/or data sets from the real world. A metaheuristic is a higher-level procedure or heuristic that is designed to find, generate, or select a heuristic that may provide a sufficiently good solution to an optimization problem. Similar to heuristics, metaheuristics do not guarantee that a globally optimal solution will be found. However, since they may make relatively few assumptions with regard to the problem under consideration, they are, in practice, often used for a large variety of optimization problems. Metaheuristics include genetic algorithms, simulated annealing, and tabu search.

3 Scheduling in Manufacturing

Clearly there are a plethora of manufacturing environments where scheduling plays an important role. In this section, we consider two such environments, namely, job shop scheduling environments and resource constrained project scheduling environments. An example of the first one would be a semiconductor manufacturing facility and an example of the second one would be a shipyard that builds large cruise ships.

3.1 Job Shop Scheduling

A job shop typically consists of a number of machines that may be configured in any way, providing many different machine configurations to be considered. The simplest machine environment is clearly the single machine. An enormous amount of research has been done on single machine scheduling, see Abdul-Razaq et al. (1990), Koumas (2010), and Adamu and Adewumi (2014), for example.

In this subsection, three classes of parallel machine scheduling problems are considered—basic problems, more general problems, and more practical problems. In the basic problem category, two types of objectives are of interest: min-sum objectives (e.g., the total weighted completion time and the total completion time) in Sections 3.1.1 and 3.1.2 and min-max objective (e.g., the makespan) in Section 3.1.3. In the category of more general problems, additional features such as due dates and release dates play a role as well in Section 3.1.4. Finally, a job shop scheduling is introduced along with a practically efficient solution methodology called the Shifting Bottleneck Heuristic in Section 3.1.5.

3.1.1 Scheduling Problems with the Total Weighted Completion Time Objective

Recall that for the single machine problem with the total weighted completion time objective, the WSPT rule is optimal. This problem is thus tractable. An immediate generalization of the single machine configuration is a configuration with \( m \) identical machines in parallel. Consider \( m \) machines in parallel and \( n \) jobs. A job can be processed on any one of the \( m \) machines, and the processing time of job \( j \) is \( p_{ij} \) on any one of the machines. The objective is still the minimization of the total weighted completion time. Obviously, \( m \) machines in parallel is a more general setting
than a single machine. This parallel machine scheduling problem with the total weighted completion time objective is known to be NP-Hard (Bruno et al. 1974), implying that the WSPT rule may not yield an optimal schedule. However, it is of course still possible to use the WSPT rule as an approximation scheme for the parallel machine case.

An application of the WSPT rule in a parallel machine environment can be described as follows: Sort the jobs in decreasing order of their WSPT ratio and assign them one after another to the machine that becomes available the earliest. In a schedule generated by the WSPT rule, all jobs assigned to the same machine follow one another in WSPT order. Thus, the WSPT rule is a reasonable approach for the parallel machine case even though it does not guarantee an optimal solution. An algorithm for a minimization problem is called a $\rho$-approximation algorithm if the objective function value of the schedule generated by the algorithm is at most $\rho$ times the optimal objective function, and the value $\rho$ is called the worst-case performance ratio. Indeed, it has been shown that the worst-case performance ratio of the WSPT rule is no more than $(1 + \sqrt{2}) / 2 \approx 1.21$ (Kawaguchi and Kyan 1986).

Now consider a more general case where the processing time of a job depends on the machine on which the job is processed. In other words, if job $j$ is processed on machine $i$, its processing time is $p_{ij}$ time units. If the processing time of a job depends on both the job and the machine, then the machine environment is referred to as unrelated parallel machines. Since the identical parallel machine case is NP-Hard, the unrelated parallel machine case is clearly NP-Hard as well. The WSPT rule cannot now be applied because the processing time of a job depends on the machine.

Up to this point, we have only considered scheduling problems with the total weighted completion time objective in various different machine environments. Now consider their counterparts with the total (unweighted) completion time as objective; these are clearly special cases of the problems with the total weighted completion time objective. When an unweighted problem is considered, the WSPT priority rule becomes the SPT (Shortest Processing Time) rule because the weights of all jobs are equal to one another. Since the WSPT rule is optimal for the single machine scheduling to minimize total weighted completion time, the SPT rule is optimal for the single machine with the total completion time as objective.

### 3.1.2 Scheduling Problems with the Total Completion Time Objective

Then, what is the complexity of the problem with $m$ identical machines in parallel and the total completion time objective? Conway et al. (1967) proved that the SPT rule is still optimal for the problem with $m$ identical machines in parallel and the total completion time objective. What then is the complexity of the unrelated machines in parallel case? Actually, for this case the SPT rule is not even defined because the processing times vary over machines. In order to develop an algorithm for the problem with unrelated machines, either the SPT rule should be somehow generalized or a new idea needs to be developed.

The unrelated parallel machine scheduling problem with the total completion time objective can be formulated as a so-called weighted bipartite matching problem with $n$ jobs on one side and $mn$ positions on the other side (Horn 1973; Bruno et al. 1974). It can also be considered as an “assignment problem” or as a special case of the transportation problem. If job $j$ is processed on machine $i$ and there are $k - 1$ jobs following job $j$ on machine $i$, then job $j$ contributes $k \times p_{ij}$ to the value of the objective function. There are well-known algorithms that run in polynomial time for the weighted bipartite matching problem: one of the more popular algorithms is the so-called Hungarian Method (Edmonds 1965; Hopcroft and Karp 1973). The unrelated parallel machine scheduling problem with the total completion time objective can therefore be solved in polynomial time.
Figure 5.1 shows the relationships between the special cases and the more general cases of the problems described above and their solution methodologies. Between the problems, arrows go from the more specialized problems to the more general problems. For the tractable problems, their polynomial time algorithms are presented while for the intractable problems, approximation algorithms are presented with their worst-case performance ratio.

### 3.1.3 Scheduling Problems with the Makespan Objective

Now consider a machine environment in which the makespan has to be minimized. When all jobs are available at time zero, the makespan on a single machine is always equal to the total processing time. The order in which the jobs are processed does not affect the makespan.

(i) **Identical machines in parallel and the makespan**: The simplest case is the case with identical machines in parallel. Scheduling identical parallel machines with the makespan objective is known to be NP-Hard, even for two machines in parallel. We can examine two priority rules for this problem with \( n \) jobs on \( m \) machines, namely, the List Scheduling (LS) rule and the Longest Processing Time (LPT) rule. List Scheduling is straightforward. Take the set of jobs in any order and for each job, allocate this job to the machine that has currently the smallest load. The worst-case performance ratio of the LS rule is known to be \( 2 - 1/m \) (Graham 1966). The LPT rule first sorts the jobs in decreasing order of their processing times and then applies the List Scheduling rule. The worst-case performance ratio of the LPT rule is known to be \( 4/3 - 1/3m \) (Graham 1969).

(ii) **Uniform machines in parallel and the makespan**: Consider parallel machines with different speeds. In the scheduling literature, such an environment is typically referred to as “uniformly related machines” or simply “uniform machines.” In the uniform machines case, machine \( i \) has a speed \( s_i \) and job \( j \) has a processing time \( p_j \), and the time requirement (realized processing time) of job \( j \) on machine \( i \) is \( p_j / s_i \). Thus, \( p_{ij} = p_j / s_i \). However, when all \( s_i \) values are the same, then the problem reduces to identical machines in parallel. Applying LPT in a uniform machines environment may seem a reasonable approach. However, it may require a minor modification. In a uniform machines environment, a job should be tried out on every machine and then be assigned to the machine where it would have the minimum completion time. Several attempts have been made to determine the exact worst-case performance ratio of the LPT rule. (See Gonzalez et al.)

![Figure 5.1](image-url) Relationships between Problems with Total Weighted Completion Time Objective
(Dobson (1984), and Friesen (1987) for further information.) Kovács (2010) obtained the most recent results for this problem and showed that the worst-case performance ratio lies in the interval $[1.54, 1.5773]$.

(iii) Unrelated machines in parallel and the makespan: In an unrelated parallel machine environment, the processing time of job $j$ on machine $i$ is $p_{ij}$. In this case, the LPT rule cannot be applied since it is not possible to sort the jobs according to their processing times. However, the problem can be formulated as an integer programming problem:

Minimize $C_{\text{max}}$

Subject to

\[ \sum_{i=1}^{m} p_{ij} x_{ij} \leq C_{\text{max}} \quad \text{for } i = 1, \ldots, m \]

\[ \sum_{j=1}^{n} x_{ij} = 1 \quad \text{for } j = 1, \ldots, n \]

\[ x_{ij} \in \{0, 1\} \quad \text{for } i = 1, \ldots, m, \text{ for } j = 1, \ldots, n \]

Even though the problem can be formulated easily as a mathematical program and has been studied extensively, the first applicable solution methodology was developed only in 1990. In general, an integer programming problem is hard to solve optimally. Lenstra et al. (1990) proposed a polynomial time algorithm that generates a schedule with a makespan that is guaranteed to be no longer than twice the optimum. The idea is to solve its linear programming relaxation (by relaxing the integrality constraints to non-negativity constraints) and to round the fractional solution in a clever way to an integer solution. This idea has been analyzed with regard to the quality of the solution as well as with regard to its computational time by Gairing et al. (2007), Shchepin and Vakhania (2005), and Arad et al. (2014).

In summary, the LPT rule as developed for the case of identical machines in parallel can be extended to the uniform machine case; however, it cannot be extended to the unrelated machines case.

3.1.4 Job Shop Scheduling with Additional Features

Job shop scheduling problems may have a variety of additional features. Job $j$ may have a release date (denoted by $r_j$) when job $j$ becomes available and may have a due date (denoted by $d_j$) by when job $j$ is supposed to be completed. If the jobs have due dates, the maximum lateness (denoted by $L_{\text{max}}$) is often considered an objective of interest and the Earliest Due Date first (EDD) rule, which sequences the jobs in non-decreasing order of their due dates, is a possible algorithm. For a single machine scheduling problem with the maximum lateness as objective, the EDD rule is optimal (Jackson 1955). However, if the jobs have different release dates, then the problem turns out to be NP-Hard (Lenstra et al. 1977).

If a job can be interrupted at any time and resumed at a later point in time, it is said that preemptions are allowed. Precedence constraints imposed on the jobs may be described in the form of an acyclic graph (a directed graph with no cycles) $G = (V, A)$, where $V$ is a set of nodes that represent the jobs and $A$ is a set of directed arcs that denote the precedence constraints, i.e., pair $i$ and $j$ with $(i, j) \in A$ implies that job $i$ must precede job $j$. Lageweg et al. (1976) studied various single machine scheduling problems with release and due dates, precedence constraints, and preemptions being allowed. This showed that some cases can be solved by the EDD rule. Since then, a significant amount of progress has been made, both theoretically as well as experimentally as seen in Grabowski et al. (1986), Gupta and Kyparisis (1987), and Uzsoy et al. (1992).
3.1.5 The Shifting Bottleneck Heuristic for Job Shop Scheduling

We will consider now a more complicated scheduling problem in a job shop environment with a practical solution approach, the so-called Shifting Bottleneck Heuristic (SBH). It is assumed that there are \( n \) jobs and \( m \) machines with each job having to follow a predetermined route (a sequence of machines) and with no recirculation. Job \( j \) comprises of a set of operations \((i,j)\) which represents the processing of job \( j \) on machine \( i \) with its processing time denoted by \( p_{ij} \).

The problem of minimizing the makespan in a job shop without recirculation can be represented by a so-called disjunctive graph. A disjunctive graph is defined as follows: A directed graph \( G = (N,A,B) \) has a set of nodes \( N \) and two sets of arcs \( A \) and \( B \). The nodes in \( N \) correspond to all operations \((i,j)\) of the \( n \) jobs. The so-called conjunctive (solid) arcs in set \( A \) represent the routes of the jobs. If arc \((i,j) \rightarrow (h,j)\) is part of \( A \), then operation \((i,j)\) precedes operation \((h,j)\). Two operations from different jobs that have to be processed on the same machine are connected to one another by two so-called disjunctive (dotted) arcs going in opposite directions; these disjunctive arcs belong to set \( B \). All arcs from a node, conjunctive as well as disjunctive, have as length the processing time of the operation that is represented by that node. In addition, there is a source node \( U \) and a sink node \( V \), which are dummy nodes. The source node \( U \) has \( n \) conjunctive arcs going to the first operations of the \( n \) jobs and the sink node \( V \) has \( n \) conjunctive arcs coming in representing all the last operations. The arcs from the source have length zero, see Figure 5.2.

A feasible schedule corresponds to a selection of one disjunctive arc from each pair of disjunctive arcs such that the resulting directed graph is acyclic. Such a selection determines a unique sequence of all the operations on each machine. It is known that the problem of minimizing the makespan is reduced to finding a selection of disjunctive arcs that minimizes the length of the longest path (i.e., the critical path) in the graph. However, since this problem has turned out to be very hard, many heuristic procedures have been developed. One of the best-known procedures is the Shifting Bottleneck Heuristic (SBH) developed by Adams et al. (1988).

The basic purpose of SBH is to determine the sequence of operations on the machines, machine by machine (Pinedo 2009). Let \( M \) denote the set of all machines, and let \( M_0 \) denote the set of machines whose operations already have a fixed sequence. In an iteration, a machine from \( M - M_0 \) is then chosen to be included in \( M_0 \). The graph has all conjunctive arcs and the fixed disjunctive arcs associated with \( M_0 \) (where \( M_0 \) initially is an empty set). With this graph, we can do in an iteration a forward and a backward pass using the Critical Path Method (CPM) and get for each operation its earliest possible starting time and its latest possible finishing time in order to ensure that the makespan is not going to be increased. Then, we consider for each machine in \( M - M_0 \) a single machine scheduling problem with the maximum lateness (\( L_{\text{max}} \)) objective where each operation’s release date and due date are each operation’s earliest starting time and latest finishing time, respectively. As stated previously, this single machine scheduling problem is NP-Hard; however, procedures have been developed that perform reasonably well in practice. The minimum \( L_{\text{max}} \) of such a single machine problem corresponding to a machine is a measure of the criticality of that machine. After solving all these single machine problems, the machine in \( M - M_0 \) with the largest maximum lateness, say machine \( h \), is chosen in this iteration to be included next in set \( M_0 \) because this machine is in a sense the most critical, or the “bottleneck.”

If the disjunctive arcs that specify the sequence of operations on machine \( h \) are inserted in the graph, then the makespan of the current partial schedule is updated. In the next iteration, the entire procedure is repeated, and another machine is added to the current set \( M_0 \cup \{h\} \). The procedure stops when all machines are in \( M_0 \) or the computed \( L_{\text{max}} \) is non-positive. Figure 5.2 depicts an example of a problem instance with three jobs and four machines in its disjunctive graph.
Table 5.1 An Example of a Problem Instance for Job Shop Scheduling

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Machine Sequence</th>
<th>Processing Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1, 2, 3</td>
<td>$p_{11} = 10$, $p_{21} = 8$, $p_{31} = 4$</td>
</tr>
<tr>
<td>2</td>
<td>2, 1, 4, 3</td>
<td>$p_{22} = 8$, $p_{12} = 3$, $p_{42} = 5$, $p_{32} = 6$</td>
</tr>
<tr>
<td>3</td>
<td>1, 2, 4</td>
<td>$p_{13} = 4$, $p_{23} = 7$, $p_{43} = 3$</td>
</tr>
</tbody>
</table>

Figure 5.2 Disjunctive Graph for Job Shop with the Makespan as the Objective

This SBH idea can be extended to various other scheduling problems. For an example, see the problem of minimizing the total weighted tardiness in a job shop in Pinedo and Singer (1999).

3.2 Resource Constrained Project Scheduling

Resource constrained project scheduling fits within a framework of machine scheduling and also within a framework of project management. This type of problem, typically referred to as a Resource Constrained Project Scheduling Problem (RCPSP), has received an enormous amount of research attention over the years (as seen in Kolisch (1995) and Demeulemeester and Herroelen (2002) where a variety of formulations and solution methodologies for RCPSP are discussed).

RCPSP is one of the hardest scheduling problems around. However, some special cases can be solved in polynomial time. The most critical aspects of a project scheduling problem are its precedence constraints. The precedence relationships are expressed by an activity-on-node graph $G = (V, A)$ where $V$ and $A$ stand for the set of activities and the set of precedence relationships, respectively. In addition, the RCPSP is also subject to resource constraints. If we only consider precedence constraints and no resource constraints, the problem is actually easy.

A well-known method, referred to as the Critical Path Method (CPM), minimizes the total project duration (makespan) in project scheduling without resource constraints. This particular problem can actually also be formulated as a Linear Programming problem; a generalization of this problem can even be formulated as a Linear Programming problem. In project management, it is quite common for a project to be behind schedule or that the deadline for its completion has been moved earlier. Shortening the duration of a project is often called project crashing. In such a case, besides the makespan objective, there may be multiple objectives since crashing costs now have to be taken into consideration as well. The original processing time of job $j$ is denoted by $p_j^{\text{max}}$ and its lower bound of crashed processing time is denoted by $p_j^{\text{min}}$. If the processing
time is shortened by \( x_j \), the crashing cost is \( c_j x_j \). In some cases, there is a due date for each job (a milestone), and if job \( j \) cannot be completed on or before its due date \( d_j \), a tardiness penalty is incurred equal to \( w_j T_j \) where \( w_j \) is a unit tardiness penalty and \( T_j = \max\{0, C_j - d_j\} \) is the tardiness of job \( j \). The overall objective is now to minimize the sum of three objectives, namely, the total weighted tardiness, the total crashed cost, and the makespan. This problem can be formulated as the following Linear Program problem:

\[
\begin{align*}
\text{Minimize} & \quad \sum_{j=1}^n w_j T_j + \sum_{j=1}^n c_j x_j + C_{\text{max}} \\
\text{Subject to} & \quad S_i + p_i \leq S_k & \text{for } (j, k) \in A \\
& \quad p_j \geq p_j^{\text{min}} & \text{for } j \in V \\
& \quad p_j = p_j^{\text{max}} - x_j & \text{for } j \in V \\
& \quad T_j \geq (S_j + p_j) - d_j & \text{for } j \in V \\
& \quad t_j \geq 0, x_j \geq 0, S_j \geq 0 & \text{for } j \in V
\end{align*}
\]

Therefore, project scheduling with linear crashing costs can be formulated as a Linear Programming problem and can thus be solved in polynomial time.

4 Scheduling in Services

This section focuses on two scheduling paradigms that are important in a variety of different service industries, namely, personnel scheduling (which is often also referred to as shift scheduling) and appointment scheduling. Personnel scheduling is, for example, very important in the scheduling of operators in call centers (which play an important role in financial services, transportation industries, and so on). The second paradigm considered is appointment scheduling, which is very important in the health care industry.

4.1 Personnel Scheduling

Personnel scheduling is very important in many service industries, since schedules have to be created in such a way that at all times operators will be available to meet a fluctuating and random demand, without causing too much queueing. This is very much in contrast with personnel scheduling in manufacturing, which is inherently much easier and less challenging. An enormous amount of research has been done on personnel scheduling. This has resulted in numerous survey papers and books by authors such as Tien and Kamiyama (1982), Burgess and Busby (1992), Nanda and Browne (1992), and Burke et al. (2004).

4.1.1 Mathematical Formulation

Consider the most basic personnel scheduling problem with cycles being fixed in advance. In certain settings a cycle may be a week, while in others it may be a day or a month. A predetermined cycle consists of \( m \) time intervals or periods. When scheduling operators in call centers, a cycle typically is a day with an interval or period being usually 15 minutes (however, the lengths of the periods within a cycle do not necessarily have to be all identical). During period
$i, i = 1, \ldots, m$, the presence of $b_i$ personnel is required in order to meet anticipated (forecast) demand. The number $b_i$ is, of course, an integer. Each work assignment pattern (type of shift) in a cycle has its own cost and the objective is to minimize the total cost of all the personnel assigned to the different shift patterns while meeting demand at all times. The problem can be formulated as follows: There are $n$ different shift patterns and each employee is assigned to one and only one pattern. Shift pattern $j$ is defined by a vector $(a_{ij}, a_{2j}, \ldots, a_{mj})$. The value $a_{ij}$ is either 0 or 1; it is a 1 if period $i$ is a work period for this shift pattern and 0 otherwise. Let $c_j$ denote the cost of assigning a person to shift $j$ and $x_j$ the (integer) decision variable representing the number of people assigned to shift pattern $j$. The problem with minimizing the total cost of assigning personnel to meet demand can be formulated as the following integer programming problem:

Minimize 
\[ \sum_{i=1}^{n} c_i x_i + c \sum_{i=1}^{m} x_i + \sum_{i=1}^{n} c x_i \]

Subject to 
\[ a_{ij} x_1 + a_{2j} x_2 + \cdots + a_{mj} x_m \geq b_i \]
\[ a_{ij} x_1 + a_{2j} x_2 + \cdots + a_{mj} x_m \geq b_j \]
\[ \vdots \]
\[ a_{ij} x_1 + a_{2j} x_2 + \cdots + a_{mj} x_m \geq b_n \]
\[ x_j \geq 0 \quad \text{for} \quad j = 1, \ldots, n, \quad \text{with} \quad x_1, \ldots, x_n \text{ integer.} \]

In matrix form, this integer programming problem can be written as follows:

Minimize 
\[ \vec{c} \vec{x} \]

Subject to 
\[ \mathbf{A} \vec{x} \geq \vec{b} \]
\[ \vec{x} \text{ integer} \geq 0. \]

Such an integer programming problem in its most general form is known to be NP-Hard.

### 4.1.2 Impact of Breaks

However, the $\mathbf{A}$ matrix in practice may often exhibit a special structure. For example, shift $j$, $(a_{ij}, \ldots, a_{mj})$, may contain a contiguous set of 1’s (a contiguous set of 1’s implies that there are no 0’s in between 1’s). However, the number of 1’s may often vary from shift to shift, since it is possible that some shifts have to work longer hours (or more days) than other shifts. Even though the integer programming formulation of the general personnel scheduling problem (with an arbitrary 0-1 $\mathbf{A}$ matrix) is NP-Hard, the special case with each column containing a contiguous set of 1’s is easy. It can be shown that the solution of the linear programming relaxation is then always an integer. There are several other important special cases that are solvable in polynomial time.

However, in practice, the columns in the integer programming problem described above do not always contain a contiguous set of ones. The operators are often entitled to breaks (e.g., lunch breaks and coffee breaks). Such breaks imply that a column may have some zeroes sprinkled in between the ones. An integer programming problem with an arbitrary number of zeroes sprinkled in between the ones is known to be NP-Hard. However, if each particular shift has only one break, then it has been shown that the problem still can be solved in polynomial time. In the case that a shift has a lunch break as well as two coffee breaks, then a heuristic is often used: as a first step, the breaks are ignored. This implies that the columns in the integer programming
problem have contiguous sets of ones. A solution is then obtained via Linear Programming. Since the breaks imply that in any period in actuality fewer operators will be available to serve the customers, the $b_i$ vector of the Linear Programming problem is adjusted upwards to take this into account. A solution of the Linear Programming problem specifies the number of people that have to be hired for each shift type. After a solution has been generated via Linear Programming, the breaks are inserted through a straightforward break insertion heuristic.

Many papers have focused on a number of special cases of the problem described above; see Bartholdi et al. (1980), Burns and Carter (1985), Burns and Koop (1987), Emmons (1985), Emmons and Burns (1991), and Hung and Emmons (1993), for example.

4.2 Appointment Scheduling

Scheduling appointments is a common practice in many service industries, mainly in order to utilize resources efficiently and to avoid queueing. Many papers have appeared in the literature on appointment scheduling, mostly motivated by health care applications. Cayirli and Veral (2003) and Gupta and Denton (2008) provide overviews of the literature, research challenges, and opportunities. Hall (2012) provides a comprehensive review of models and methods used for scheduling the delivery of patient care in all parts of the health care system.

Outpatient clinics typically start empty at the beginning of a working day, operate for a finite amount of time (in the order 8–12 hours), and then shut down at the end of the day. An analysis of such an environment can be done through a variety of approaches, including stochastic programming, queueing theory, and stylized scheduling models.

Appointment scheduling systems are widely used as a tool for managing patient arrivals at health care facilities in order to match supply with demand. In practice, it is actually fairly common for patients not to show up for their scheduled services. Such a patient is then referred to as a no-show. No-shows result in under-utilization of valuable resources and limit the access for other patients who could have filled the empty slots. Meanwhile, patients nationwide experience difficulties in accessing medical appointments in a timely manner due to long backlogs.

4.2.1 Appointment Overbooking

Appointment overbooking is one operational strategy employed by health care providers to address the issue of no-shows and at the same time increase patients’ access to care. However, overbooking may potentially result in an overcrowded facility that increases patients’ waiting times and the system’s overtime. Recent studies have demonstrated that a sensible practice of appointment overbooking can significantly improve the operational performance of a medical facility with patients experiencing shorter waiting times and better access to services. See LaGanga and Lawrence (2012), Robinson and Chen (2010), Zacharias and Pinedo (2014a), Zacharias and Pinedo (2014b), for example.

In a setting with patients that have similar characteristics, it is of interest to determine the number of patients to schedule every day and how to allocate these patients to the different time slots in a day. The sequencing of the patients is also of interest when patients have different characteristics (different no-show probabilities, different processing times, and different waiting costs). In most cases, finding an optimal schedule is analytically intractable. Thus, most of the literature uses enumeration, search algorithms, simulation-based techniques, and/or heuristics. For a formulation of an appointment scheduling model, see Zacharias and Pinedo (2014a; 2014b).
4.2.2 Characteristics of Optimal Appointment Schedules

Optimal schedules with overbooking tend to be front-loaded: More patients are scheduled towards the beginning of the working day (in order to get an empty system to start up), and the schedules tend to become somewhat less dense towards the end of the working day (clearly in order to avoid high overtime costs). Optimal schedules may therefore exhibit three phases: A start-up phase in which the overbooking level is above average (in order to ensure that the system will be able to start up without too much idle time in the beginning of the schedule), an intermediate phase with periodic overbookings, and an emptying out phase in which the overbooking level is below the average overbooking level.

For a more comprehensive numerical analysis, the reader is referred to Zacharias and Pinedo (2014b). It is evident, and intuitive, that the optimal overbooking level is increasing in the no-show rate. As the cost of waiting increases, the optimal schedules become less front-loaded, without necessarily observing a decrease in the overbooking level. Overbooking increases significantly with the number of parallel servers. That increase is more prevalent for higher no-show rates.

Variations of the model described above have been analyzed in various papers including Robinson and Chen (2010), LaGanga and Lawrence (2012), Zacharias and Pinedo (2014a), and Zacharias and Pinedo (2014b). Most of the appointment scheduling literature focuses on single-server models. Kaandorp and Koole (2007), Klassen and Yoogalingam (2009), Robinson and Chen (2010), and Millhiser and Veral (2015) consider the appointment scheduling problem with patients that have similar characteristics who arrive on time for their scheduled appointments, if they do show up. Begen and Queyranne (2011), Cayirli et al. (2012), LaGanga and Lawrence (2012), and Zacharias and Pinedo (2014a) account further for patient heterogeneity.

5 Managerial Implications

5.1 Design of Scheduling Systems

The implementation of a scheduling process in practice typically requires the development of a computerized system. Such a scheduling system must rely on and interact with a reliable database system that contains all the information regarding the orders on file (either customers or jobs) and the machine availability times. A scheduling system must also have at its disposal a scheduling engine that has a library of scheduling algorithms at its disposal, which includes priority rules and heuristic techniques (e.g., local search procedures, including simulated annealing and tabu search, which are very popular in practice). For a scheduling system to be effective, it typically must have also elaborate user interfaces that facilitate interactive optimization. For an example of a Gantt chart user interface, see Figure 5.3, which depicts the Gantt chart interface of the LEKIN system, an educational scheduling system developed at the Stern School of Business, New York University (NYU). Such an interface typically has drag-and-drop capabilities. These capabilities allow the scheduler to manually adjust a schedule that had been generated by the scheduling engine and check the feasibility of the revised schedule easily. Figure 5.4 depicts the user interface of a surgery scheduling system for a hospital.

In practice, any scheduling system must allow for a user-friendly way of manipulating an already existing schedule. There are many reasons why in practice existing schedules often have to be rescheduled. Usually, this is due to the occurrence of a random event, such as a machine breakdown, a sudden arrival of a high-priority job, and so on. However, there is a strong desirability that the new schedule is not too different from the already existing schedule (in order to avoid confusion).
Figure 5.3  Interface of a Job Shop Scheduling System for a Manufacturing Application

Figure 5.4  Interface of an Appointment Scheduling System (Operating Rooms in a Hospital)
5.2 Dealing with Randomness

In practice, there are many sources of randomness. Typically, processing times are not fixed but are rather random variables of which the distribution may or may not be known. Machines may be subject to breakdowns, which may make their availability periods random. New jobs or new appointments may come in at random and may have to be inserted in the schedule at once. In many scheduling environments, such random events may require frequent rescheduling. Since random events can have in practice a major impact on scheduling processes in many scheduling environments, it may not make sense to have an elaborate algorithm in place to construct a basic schedule when it is expected that the schedule will have to be changed soon afterwards. So even though the problem is very hard, it would not make that much sense to have an elaborate algorithm in place.

Thus, even if a deterministic version of a scheduling problem (assuming no randomness) is NP-Hard, a scheduler may use in practice only very simple and basic rules to generate a schedule. That is to say, he may use just a simple priority rule, even though the priority rule would not be optimal for the deterministic version of the problem. The priority rule may actually be theoretically optimal for certain stochastic counterparts of the problem.

Besides minimizing an objective function that is based on the completion times of the jobs, the scheduler may also want to maximize the robustness of the schedule. A schedule is robust if the necessary rescheduling triggered by a random event does not result in a schedule very different from the original schedule. There are several different measures of schedule robustness. For example, makespan variability, post-disturbance makespan, and various slack related functions are used in the literature.

There are several methods to make schedules more robust. A standard way is through the insertion of idle times (at times also referred to as buffer times), since scheduled idle times would mitigate the effects of machine breakdowns, unexpected arrivals of rush jobs, and so on.

6 Conclusions and Future Research Directions

Scheduling problems are clearly ubiquitous. In this chapter, we have only considered a sample of the many different types of scheduling problems that appear in practice. There are clearly many other types of scheduling problems.

For example, in manufacturing industries, scheduling plays an important role in assembly lines such as those used in the automotive industries. Assembly line scheduling has its own characteristics and requires specialized techniques. Scheduling plays also an important role in managing equipment maintenance activities; see Ait-Kadia et al. (2011), for example. Industrial robots that are designed to do a series of automated tasks require intricate scheduling techniques, see Dawande et al. (2007).

In the service industries, there are many other scheduling applications as well. This is clearly the case in transportation industries. Aircraft scheduling plays a very important role in the aviation industry (see Barnhart et al. (1998), Barnhart et al. (2003), Barnhart and Laporte (2006), Desaulniers et al. (1997), and Stojkovic et al. (2002)). In marine transportation, tanker scheduling is very important for the major oil companies, see Christiansen et al. (2004), Christiansen et al. (2006), and Perakis and Bremer (1992). Scheduling in public transport (e.g., train and bus scheduling) has received an enormous attention as well, that we see in works by Daduna et al. (1995) and Voss and Daduna (2001). In the entertainment industry, tournament scheduling is of the utmost importance in the sports industries (see Aggoun and Vazacopoulos (2004), Nemhauser and Trick (1998), Schaerf (1999), and Bartsch et al. (2006)). At universities, classroom...
scheduling and exam scheduling are very important (e.g., Burke et al. (1996)). Each one of the scheduling applications mentioned above has its own idiosyncrasies and leads to very specific problem formulations.

There are many avenues for future research in scheduling. Combinatorial optimization models, online scheduling models, stochastic scheduling models, as well as robust optimization applications will continue to receive significant research attention.

References and Bibliography


Kangbok Lee and Michael Pinedo
Scheduling in Manufacturing and Services


