# CHAPTER 63
# Scheduling and Dispatching

**MICHAEL PINEDO AND SRIDHAR SESHADRI**
New York University

## 1. INTRODUCTION

Detailed scheduling of the various elements of a production system is crucial to the efficiency and control of operations. Orders have to be released and have to be translated into one or more jobs with associated due dates. The jobs often have to be processed by the machines of a workcenter in a given order or sequence. Queueing may occur when jobs have to wait for processing on machines

Edited by Gavriel Salvendy Copyright © 2001 John Wiley & Sons, Inc.

that are busy; preemptions may occur when high-priority jobs arrive at busy machines and have to proceed at once.

The scheduling and dispatching process has to interface with several other functions in the organization. On the one hand, it is affected by the production planning process, which handles medium- and long-term planning for the entire organization. Production planning takes inventory levels, forecasts, and resource requirements into account in order to do some form of optimization at a higher level. Any decision taken by this planning function may have an impact on scheduling and dispatching. On the other hand, scheduling also receives input from shop-floor control. Events that happen on the shop-floor have to be taken into account because they may have a considerable impact on the schedules.

What follows mainly focuses on the detailed scheduling of the jobs. Given a collection of jobs that have to be processed in a given machine environment, the problem is to schedule the jobs, subject to given constraints, in such a way that one or more performance criteria are optimized. Various forms of uncertainties, such as random job-processing times, machines subject to breakdown, and rush orders, may have to be dealt with.

This chapter is organized as follows. Section 2 presents some general notation as well as a mathematical framework that is used in the theory of scheduling and dispatching. The subsequent sections focus first on dispatching techniques and then on scheduling procedures. There are two reasons for following this approach: first, dispatching rules are typically easier to explain than scheduling procedures, and second, scheduling procedures often use dispatching rules as subroutines within a more elaborate framework. Understanding the workings of a scheduling procedure thus typically requires a certain knowledge of dispatching rules. Section 3 therefore gives an overview of the theory that has been developed with regard to dispatching. It covers basic dispatching rules as well as composite dispatching rules. Section 4 subsequently focuses on the main techniques applied to scheduling; it also describes a number of empirical procedures that have proven to be useful in current scheduling systems. Section 5 focuses on scheduling and dispatching problems in practice; it discusses a variety of industrial environments where scheduling is of critical importance. Section 6 first discusses the general structure of scheduling systems and then gives a description of the major trends in the development of industrial scheduling systems during the last two decades. Section 7 contains a discussion of the difficulties encountered in the implementation of these systems (see Figure 1).

Sections 3 and 4 are somewhat technical; 5 and 6 are more descriptive and basically self-contained. The less technically inclined reader may prefer to read these last two sections first.

## 2. FRAMEWORK AND MODELS

### 2.1. Models and Notation

In the past four decades, a significant amount of theoretical research has been done in scheduling as well as in dispatching. Along the way, a notation has evolved that succinctly captures the structure of most machine scheduling models studied in the literature. A short description of this framework and notation is presented here (see Lawler et al. 1989; Pinedo 1995).

The number of jobs is denoted by $n$ and the number of machines by $m$. The subscript $j$ refers to a job, while the subscript $i$ refers to a machine. The following data are associated with job $j$:

$p_{ij}$ = the processing time of job $j$ on machine $i$; if job $j$ is only to be processed on one machine or if it has the same processing times on each one of the machines it has (or is allowed) to be processed on, the subscript $i$ is omitted.
$r_j$ = release date of job $j$.
$d_j$ = the due date of job $j$.
$w_j$ = the weight (importance) of job $j$.

A sequencing or scheduling problem is described by a triplet $\alpha|\beta|\gamma$, where $\alpha$ describes the machine environment, $\beta$ the processing characteristics and constraints, and $\gamma$ the objective to be minimized. Examples of the possible (exclusive) entries in the $\alpha$-field are:

1 = a single machine.
*Pm* = $m$ identical machines in parallel; a job may be processed on any one of the $m$ machines, it does not matter which one.
*Fm* = a flow shop of $m$ machines; that is, $m$ machines in series. A job after completion at one machine joins the queue at the next machine. All queues operate under the first-in-first-out discipline, that is, a job cannot "pass" another while waiting in a queue.
*Jm* = a job shop of $m$ machines; in such a shop each job has its own route through the various machines and a job may visit a machine more than once.

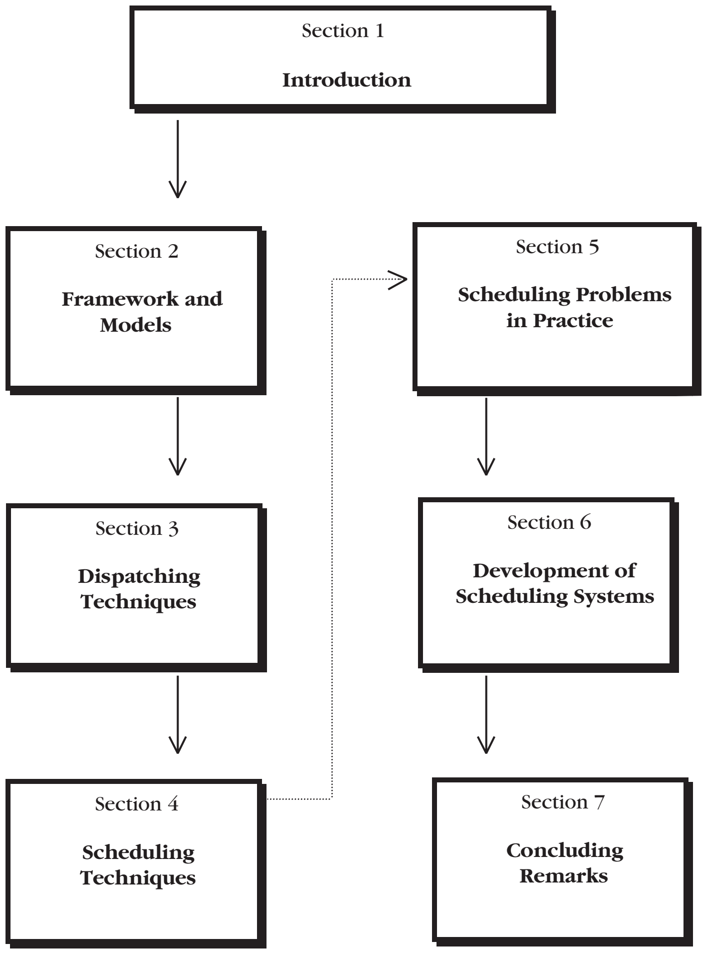


**Figure 1** Outline of This Chapter.

Examples of entries in the $\beta$-field are:

| | |
|---|---|
| $r_j$ = | the release date of job $j$; job $j$ may not start its processing before this date. If no $r_j$ appears in this field, all $r_j$ are assumed to be *0*. |
| $s_{jk}$ = | the sequence dependent setup time between jobs $j$ and $k$; $s_{0k}$ denotes the setup time for job $k$ in case job $k$ is first in the sequence, while $s_{j0}$ denotes the clean-up time after job $j$ in case job $j$ is last in the sequence (of course, $s_{0k}$ and $s_{j0}$ may be zero). If no $s_{jk}$ appears in the $\beta$-field, all setup times are assumed to be 0. |
| *prec* = | the precedence constraints among the jobs. If no *prec* appears, there are no precedence constraints. |
| *prmp* = | preemptions are allowed. If *prmp* does not appear, preemptions are not allowed. |

Most other entries that may appear in the $\beta$-field are self-explanatory. The $\beta$-field may have multiple entries or may be completely empty. Due dates, in contrast with release dates, are not specified in this field; the ojective implies whether or not the jobs have due dates.

The objective is always a function of the completion times of the jobs, which are, of course, schedule dependent. The completion time of job $j$ is denoted by $C_j$. Examples of possible objective functions to be minimized are:

| | | |
|---|---|---|
| $C_{max}$ | = | the makespan, defined as $\max(C_1, \ldots, C_n)$, which 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). |
| $L_{max}$ | = | the maximum lateness, which is defined as $\max(L_1, \ldots, L_n)$, where $L_j$ equals $C_j - d_j$. |
| $\Sigma w_j C_j$ | = | the sum of the weighted completion times of the $n$ jobs. |
| $\Sigma w_j T_j$ | = | the sum of the weighted tardinesses, where the tardiness of job $j$, $T_j$, is defined as $\max(C_j - d_j, 0)$. |
| $\Sigma w_j U_j$ | = | the weighted number of tardy jobs, with $U_j$ being 1 if $C_j \geq d_j$ and 0 otherwise. |

The following examples illustrate the notation:

$1|s_{jk}|C_{\max}$ denotes a single machine with $n$ jobs subject to sequence-dependent setup times; the objective is to minimize the makespan. It is well known that this problem is equivalent to the so-called traveling salesman problem in which a salesman has to tour $n$ cities in such a way that the total distance traveled is minimized.

$Pm|r_j, s_{jk}|\Sigma w_j T_j$ denotes $m$ identical machines in parallel, $n$ jobs with different release dates, different due dates, and different weights. The jobs are subject to sequence-dependent setup times and the objective is to find a sequence that minimizes the sum of the weighted tardinesses.

$Fm|p_{ij} = p_j|\Sigma w_j C_j$ denotes a so-called *proportionate* flow shop with $m$ machines, i.e., $m$ machines in series, with the processing times of job $j$ on all $m$ machines identical and equal to $p_j$ (which is the reason this flow shop is called 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.

$Jm||C_{max}$ denotes a job shop with $m$ machines; the objective is to minimize the makespan.

Of course, there are many scheduling models that are not captured by this framework. Many situations in the real world have machine environments that do not fit the framework. For example, various researchers have recently begun to study hybrids between flow shops and parallel machines. In one such hybrid, there are a number of stages in series, with each stage consisting of a number of machines in parallel. Jobs progress from stage to stage, and at each stage each job has to be processed on one of the parallel machines. This machine environment has been given various names in recent years: generalized flow shop, flexible flow shop, compound flow shop. Alternatively, one can consider a number of parallel flow shops, each having a single machine at every stage. When a job is assigned to a particular flow shop, it is not allowed to switch in the middle of the process to another flow shop (see Figure 2).

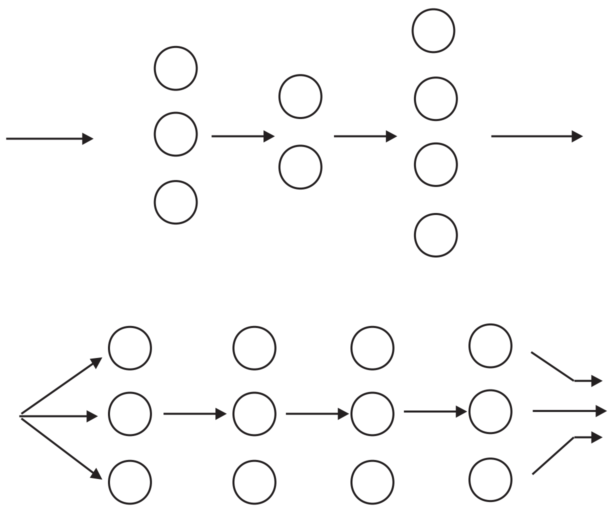


**Figure 2** Hybrids of Flow Shops and Parallel Machines.

All objective functions mentioned above are so-called regular performance measures. Regular performance measures are functions that are *nondecreasing* in $C_1, \ldots, C_n$. Recently researchers have begun to pay attention to objective functions that are not regular. For example, when job $j$ has due date $d_j$, there may be, besides a tardiness penalty, also an earliness penalty; the larger the deviation from the due date, in either direction, the larger the penalty. The objective may be the sum of earliness penalties and tardiness penalties.

The framework has been primarily designed for models with a single objective because most of the research has been concentrated on such models. Currently, researchers are studying models with multiple objectives as well, but a standard notation with regard to such models has not been developed yet.

Various other features in scheduling, not mentioned above, have been studied and analyzed in the past, such as finite buffers, blocking, and recirculation. A standard notation for these features still needs to be developed.

## 2.2. Overview of Past Research Directions

The research in deterministic scheduling problems has followed a number of different directions (see Pinedo 1995). Three areas have received considerable attention:

### *2.2.1. Determining Computational Complexity*

For some problems so-called polynomial time algorithms are known to exist. A polynomial time algorithm implies that the number of computational steps (which is proportional to the amount of computer time) needed to find a schedule which achieves the optimum value of the objective function is a polynomial function of the parameters of the problem (e.g., the number of jobs, $n$ and/or the number of machines, $m$). A polynomial time algorithm may require, for example, a number of steps that is on the order of $n^3$ or $n^4$. There are problems, however, for which no polynomial time algorithm is known to exist. These problems are the so-called NP-hard problems. The most efficient algorithms for these problems are exponential in the parameters of the problems. Such algorithms may require, for example, a number of steps that is on the order of $3^n$ or $4^n$.

Determining whether or not a scheduling problem can be solved in polynomial time typically involves proving that the given problem is in some sense equivalent to a problem already known to be NP-hard. For example, the $1|s_{jk}|C_{max}$ problem is NP-hard because it is equivalent to the traveling salesman problem, which is known to be NP-hard.

### *2.2.2. Development of (Efficient) Algorithms*

For problems that are solvable in polynomial time, it is usually relatively easy to find efficient algorithms. For the simplest of these problems, a simple sort is all that is required. For example, it may only be necessary to order the jobs in increasing order of their due dates (the so-called earliest due date rule [EDD]) or in decreasing order of the $w_j/p_j$ ratios (the so-called weighted shortest processing time [WSPT] first rule. For more complicated polynomial time problems, more sophisticated techniques such as dynamic programming are required.

For the NP-hard problems, it is always significantly harder to find a good algorithm. One usually resorts to heuristics, which may give reasonably good schedules. When a more accurate solution is required, a more sophisticated technique (which also uses more computer time) such as branch and bound or lagrangean relaxation is usually employed. When only the optimal solution will do, the most likely approach would be to develop an (exponential time) algorithm based on dynamic programming or branch and bound.

### *2.2.3. Worst-Case Analysis of Heuristics*

Solutions that are optimal for the easier problems often turn out to be good heuristics for the more complicated NP-hard problems. Examples of simple heuristics are the earliest due date (EDD) rule, the weighted shortest processing time first (WSPT) rule, the longest processing time first (LPT) rule, the shortest setup time first (SST) rule, and the critical path (CP) rule. The SST rule is of importance when the jobs are subject to sequence-dependent setup times; following the completion of a job, this rule consistently selects as the next job the one with the smallest setup time. The critical path (CP) rule is of importance when the jobs are subject to precedence constraints; following the completion of a job, this rule always selects as the next job the job that is at the head of the chain of jobs (one having to follow another) that contains the largest amount of processing.

It is of interest to know the worst-case behavior of such heuristics when applied on an NP-hard scheduling problem; that is, to have an upper bound on the ratio of the value of the objective function obtained via the heuristic divided by the true optimal value. This upper bound often, but not always, lies between 1 and 2. Besides giving an indication of how bad the result of a given heuristic may turn out to be, a worst-case analysis also gives an indication of the types of problem data for which the heuristic does not work well.

The next two sections deal, in particular, with heuristics and algorithms. At times a problem may be said to be NP-hard; however, no details will be given on how this is determined.

## 3. DISPATCHING TECHNIQUES

### 3.1. Basic Dispatching Rules

The five priority rules mentioned in the previous section, WSPT, EDD, LPT, SST, and CP, are fairly important. They provide optimal sequences in some very simple cases and serve as heuristics for more complicated scheduling models. It is useful to know the properties of these priority rules when designing a complicated computer-based scheduling system. Different modules in such a system may use at given times one of these rules to sequence a subset of the jobs. Or a composite priority rule may be constructed by combining two or more of these simple priority rules in order to minimize a mixture of various objectives. A more in-depth discussion of these five simple priority rules follows.

#### *3.1.1. The WSPT Rule*

The weighted shortest processing time first rule, which schedules the jobs in decreasing order of $w_j/p_j$, actually minimizes the sum of the weighted completion times on a single machine (see Smith 1956), that is, $1||\Sigma w_jC_j$. However, for slightly more complicated problems, such as $1|r_j|\Sigma w_jC_j$, or $1|r_j,prmp|\Sigma w_jC_j$, or $Pm||\Sigma w_jC_j$, the WSPT rule (or any variant of the WSPT rule that takes preemptions into account) does not necessarily result in the optimal solution. These three problems are actually NP-hard. The WSPT rule is nevertheless a very good heuristic. A worst-case analysis shows that for $Pm||\Sigma w_jC_j$:

$$\frac{\Sigma w_jC_j(\text{WSPT})}{\Sigma w_jC_j(\text{OPT})} \leq \frac{1+\sqrt{2}}{2} = 1.207$$

where $\Sigma w_jC_j(\text{WSPT})$ ($\Sigma w_jC_j(\text{OPT})$) denotes the value of the objective function under the WSPT (OPT) rule (see Kawaguchi and Kyan 1986). For the special case where all jobs have equal weights, that is $w_j = 1$ for $j = 1, \ldots, n$, the WSPT rule reduces to the shortest processing time first (SPT) rule. It is well known that the SPT rule minimizes the total completion time on parallel machines ($Pm||\Sigma C_j$).

#### *3.1.2. The EDD Rule*

The earliest due date rule, which schedules the jobs in increasing order of their due dates, minimizes the maximum lateness on a single machine (see Jackson 1955), that is, $1||L_{max}$, as well as in a proportionate flow shop, that is, $Fm|p_{ij} = p_j|L_{max}$. However, it does not provide an optimal solution for other due date-related problems, such as $1||\Sigma T_j$. Instances of $1||\Sigma T_j$ with

$$\frac{\Sigma T_j(\text{EDD})}{\Sigma T_j(\text{OPT})} = n$$

can be found easily. This implies that the EDD rule at times may result in a solution that is far from optimal. The EDD rule is usually not used as a heuristic by itself, but rather as part of a composite heuristic (such as the so-called ATC rule, which is discussed later in this section).

#### *3.1.3. The LPT Rule*

The longest processing time first rule, which schedules the jobs in decreasing order of $p_j$, is used as a heuristic for the $Pm||C_{\max}$ problem, which is known to be NP-hard. The importance of the makespan objective lies in the fact that a small makespan is a sign of a good job balance (partition) over the various machines. In order to find a schedule with a relatively small makespan, one orders the jobs in decreasing order of their processing times and puts the jobs on the machines, whenever one is freed, in that order. A worst-case analysis of this heuristic shows that

$$\frac{C_{\max}(\text{LPT})}{C_{\max}(\text{OPT})} \leq \frac{4}{3} - \frac{1}{3m}$$

See Graham (1969). This implies that at all times this heuristic performs reasonably well. The LPT heuristic has been used in industrial scheduling systems in order to provide a reasonable balance of the workload over the different machines. After the partition has been determined by the LPT rule, the jobs assigned to any given machine can be resequenced. Resequencing a machine clearly does not affect the balance and may be done to minimize a secondary objective.

### *3.1.4. The SST Rule*

The shortest setup time first rule, when a machine is freed after completing job $j$, selects as the next job the one with the smallest setup time $s_{jk}$. It is used as a heuristic for the $1|s_{jk}|C_{\max}$ problem, which, as said above, is equivalent to the traveling salesman problem. The traveling salesman leaving a city for the city closest by (the nearest neighbor) is equivalent to sequencing jobs based on the smallest sequence-dependent setup time. The $1|s_{jk}|C_{\max}$ problem is known to be NP-hard even when the so-called *triangle inequality* holds. This inequality implies that $s_{jk} + s_{kl} \geq s_{jl}$ for all $j$, $k$, and $l$. It can be shown easily that for the $1|s_{jk}|C_{\max}$ problem, even in case the triangle inequality holds, the ratio

$$\frac{C_{\max}(\text{SST})}{C_{\max}(\text{OPT})}$$

can become arbitrarily large (see Rosenkrantz et al. 1977). The SST rule in practice is often used in composite heuristics.

### *3.1.5. The CP Rule*

The critical path rule always selects as the next job the one that is at the head of the chain of jobs that contains the largest amount of processing. It is often used as a heuristic for the $Pm|prec|C_{\max}$ problem, which is known to be NP-hard. The heuristic actually yields the minimum makespan in case all jobs have identical processing times and the precedence constraints are in the form of a *tree* (tree-like precedence constraints imply that either all jobs have at most one sucessor or all jobs have at most one predecessor). A worst-case analysis shows that for the $Pm|prec,p_j = 1|C_{\max}$ problem with arbitrary precedence constraints:

$$\frac{C_{\max}(\text{CP})}{C_{\max}(\text{OPT})} \leq 2 - \frac{1}{m-1}$$

See Chen and Liu (1975). The CP rule is used also when other objective functions have to be minimized. It plays an important role in composite heuristics as well.

## 3.2. Composite Dispatching Rules

### *3.2.1. The Apparent Tardiness Cost Heuristic*

The problem $1||\Sigma w_j T_j$ is of importance in many practical situations. Jobs frequently have different weights (priorities) as well as different due dates (committed shipping dates) with the sum of the weighted tardinesses as the objective to be minimized. This objective is strongly correlated with loss of goodwill. It is well known that this problem is NP-hard even with a single machine. So it is important to have a heuristic that provides a reasonably good schedule with little computational effort. Some heuristics come immediately to mind, namely, the WSPT rule (which is optimal when all release dates and due dates are zero) and the EDD rule (which is optimal when all due dates are sufficiently large and spread out). It is clear that a heuristic or priority rule is needed which in one form or another combines these two heuristics. The apparent tardiness cost (ATC) heuristic is such a priority rule (Vepsalainen and Morton 1987). Under this priority rule, the jobs are scheduled one by one; that is, every time the machine is freed, a priority index is computed for all the remaining jobs that are available for processing. The job with the highest priority index is then selected to go next. This priority index is a function of the time the current job completes, say $t$, as well as the $p_j$, the $w_j$, and the $d_j$ of all remaining jobs. The index is defined as

$$I_j(t) = \frac{w_j}{p_j} \exp\left(-\frac{\max(d_j - p_j - t, 0)}{K\bar{p}}\right)$$

where $K$ is a scaling parameter which is determined empirically, $\bar{p}$ is the average of the processing times of the remaining jobs, and $\max(d_j - p_j - t,0)$ is the slack of job $j$. If $K$ is chosen very large, the ATC rule reduces to the WSPT rule. On the other hand, if $K$ is chosen very close to zero, the rule reduces to the WSPT rule among the overdue jobs when there are overdue jobs; when there are no overdue jobs, it gives priority to the job with the least slack. In order to obtain good schedules, the parameter $K$ has to be chosen very carefully. This can be done by first making a detailed analysis of the particular scheduling instance under consideration. There are several ways to characterize scheduling instances. One is through a due date tightness factor $\tau$, which is defined as

$$\tau = 1 - \frac{\bar{d}}{C_{\max}}$$

where $\bar{d}$ is the average of the due dates. Another way is through a due date range factor $R$, which is defined as

$$R = \frac{d_{\max} - d_{\min}}{C_{\max}}$$

It actually pays to evaluate the $\tau$ and the $R$ of the instance under consideration and choose the scaling parameter $K$ based on these values. A significant amount of research has been done that establishes the relationship between the look-ahead parameter $K$ and the $\tau$, the $R$, and the machine environment.

Thus, when one wishes to minimize $\Sigma w_j T_j$ in a more complicated machine environment, one first characterizes the particular problem instance through a number of factors. Then one determines the value of the look-ahead parameter $K$ as a function of these characterizing factors as well as of the particular machine environment. After fixing $K$, one applies the rule.

Several generalizations of the ATC rule have been developed in order to take into account release dates as well as sequence dependent setup times. These generalizations require the initial computation of a number of factors. These factors, together with the particular machine environment, can then be used to determine a number of the parameters (see Lee et al. 1997).

Simple heuristics such as the five described above provide adequate results only for the simplest scheduling problems. Real-world scheduling problems usually require techniques that are significantly more sophisticated than a myopic priority rule that just orders the jobs according to a function of one or two parameters. There are various ways of formulating scheduling problems as well as various types of solution procedures. These are discussed next.

## 4. SCHEDULING TECHNIQUES

### 4.1. Mathematical Programming Formulations

Many scheduling problems can be formulated as linear programs or other forms of mathematical programs.

Scheduling problems that allow preemptions are often easier than problems that do not allow preemptions. The problems that allow preemptions can often be solved in polynomial time. Consider, for example, the problem $Pm|prmp|C_{\max}$. Job $j$ may be processed on any one of the $m$ machines; it may be preempted and may continue its processing on another machine at another time. A schedule that minimizes the makespan can be obtained by first solving the following linear program (LP):

$$\text{minimize} \quad C_{\max}$$

subject to

$$\begin{aligned}
\sum_{i=1}^{m} x_{ij} &= p_j, & j &= 1, \ldots, n \\
\sum_{i=1}^{m} x_{ij} &\le C_{\max}, & j &= 1, \ldots, n \\
\sum_{j=1}^{n} x_{ij} &\le C_{\max}, & i &= 1, \ldots, m \\
x_{ij} &\ge 0, & i &= 1, \ldots, m, j = 1, \ldots, n
\end{aligned}$$

The variable $x_{ij}$ represent the total time spent by job $j$ on machine $i$. The LP can be solved in polynomial time, but the solution of the LP does not prescribe an actual schedule; it merely prescribes how much time job $j$ should spend on machine $i$. However, with this information a schedule can easily (in polynomial time) be constructed. This LP formulation for the $Pm|prmp|C_{\max}$ problem is given for illustration purposes only. Actually, many schedules that minimize the makespan are easy to find and all these schedules result in a makespan

$$C_{\max} = \max(p_1, \ldots, p_n, \Sigma p_j / m).$$

One of the schedules that minimize the makespan is the preemptive longest remaining processing time first (preemptive LPT) schedule. According to this schedule, at every point in time, the $m$ jobs with the largest remaining processing times have to be processed on the $m$ machines. This schedule requires, of course, a large number of preemptions; there are other optimal schedules that do not require as many preemptions.

The formulation described above can be generalized to include the model where the processing time of a job may depend on the machine on which it is processed. Again, job $j$ needs processing on only one machine and any one will do. However, if job $j$ is processed on machine $i$ its processing time is $p_{ij}$.

Linear programming formulations are often possible either when preemptions are allowed or when all processing times are identical (i.e., $p_j = 1, j = 1, \ldots, n$). Scheduling problems with all processing times identical often reduce to the so-called assignment problem, for which there exists a linear programming formulation.

When a problem is NP-hard, a linear programming formulation is, of course, not possible. However, NP-hard scheduling problems can often be formulated as integer programs or other more complicated forms of mathematical programs. Consider, for example, the $Jm||C_{\max}$ problem. Let $p_{ij}$ denote the processing time of job $j$ on machine $i$, let $y_{ij}$ denote the starting time of this operation, and let the set $N$ denote the set of all $(i, j)$ operations. This set may be viewed as a set of nodes in a directed graph. Let the set $A$ denote the set of all precedence (routing) constraints $(i, j) \prec (k, j)$ that require job $j$ to be processed on machine $i$ before it is processed on machine $k$, that is, operation $(i, j)$ precedes operation $(k, j)$. This set may be viewed as a set of arcs in a directed graph that has nodes $N$. The following mathematical program minimizes the makespan:

$$\text{minimize } C_{\max}$$

subject to

$$\begin{aligned}
y_{kj} - y_{ij} &\geq p_{ij} && \text{for all } (i,j) \prec (k,j) \in A \\
C_{\max} - y_{ij} &\geq p_{ij} && \text{for all } (i,j) \in N \\
y_{ij} - y_{il} &\geq p_{il} \quad \text{or} \quad y_{il} - y_{ij} \geq p_{ij} && \text{for all } (i,l),(i,j), \quad i = 1, \ldots, m \\
y_{ij} &\geq 0 && \text{for all } (i,j) \in N
\end{aligned}$$

The third set of constraints is often called the disjunctive arc constraints and represent the fact that some ordering must exist among operations of different jobs that are processed on the same machine. Because of these constraints, this problem is sometimes referred to as the disjunctive programming problem.

That an NP-hard scheduling problem can be formulated as an integer programming problem or a disjunctive programming problem does not imply that there is a standard solution procedure available that will work satisfactorily. For most NP-hard scheduling problems solution procedures have to be tailor made. Two general techniques are widely used, namely dynamic programming and branch and bound. In what follows, these two techniques are applied to two well-known NP-hard problems. Dynamic programming is applied to $1||\Sigma T_j$ and branch and bound to $Fm||C_{\max}$.

## 4.2. Dynamic Programming

Dynamic programming is an optimization technique that is particularly well suited for scheduling problems with makespans that are schedule independent, such as, single-machine problems without setups, proportionate flow shops, and problems with all processing times being identical. It can also be applied on scheduling problems with makespans that do depend on the sequence.

For the $1||\Sigma T_j$ problem the *forward* dynamic programming technique is used (see Held and Karp 1962). The following observation is crucial. If the sequence $j_1, j_2, \ldots, j_n$, a given permutation of $1, \ldots, n$, is optimal for the problem under consideration, then the subsequence $j_1, j_2, \ldots, j_k$, where $k \leq n$, has to be optimal for the smaller $k$-job problem, which contains only jobs $j_1, j_2, \ldots, j_k$. Let $S$ denote a subset of the $n$ jobs and let $G(S)$ denote the minimum value of the objective function if only the jobs in $S$ are to be scheduled. It is easy to see that the recursive relationship

$$G(S) = \min_{j \in S} \left(G(S - \{j\}) + \max\left(\sum_{l \in S} p_l - d_j, 0\right)\right)$$

has to hold. The second term on the right-hand side of this expression represents the tardiness of job $j$. This recursive relationship, which in the dynamic programming literature often is referred to as the *principle of optimality,* makes it possible to solve the scheduling problem in the following manner. First, all sequencing problems that contain only a single one of the $n$ jobs are solved. There are $n$

such problems. Clearly, these problems require no optimization. However, the values of the objective functions $G(S)$ still have to be computed. If such a one-job scheduling problem consists of job $j$, then

$$G(S) = G(\{j\}) = \max(p_j - d_j, 0)$$

After these $n$ values have been computed, all sequencing problems containing two of the original $n$ jobs are solved (there are $n(n - 1)/2$ such scheduling problems); the optimal order as well as the values of the objective functions have to be determined. The recursive relationship and the results for the one-job problems make it possible to solve these two-job problems efficiently. That is, if $S = \{j, k\}$, then

$$G(\{j, k\}) = \min(G(\{k\}) + \max(p_k + p_j - d_j, 0), G(\{j\}) + \max(p_j + p_k - d_k, 0)).$$

Then all sequencing problems containing three of the $n$ jobs have to be analyzed. Again, the recursive relationship and the results for the two-job problems make it possible to solve these three-job problems efficiently. Then all four-job problems need to be analyzed, and so on.

It is easy to see that this technique is more efficient than complete enumeration. Actually, one can reduce the amount of computation even more by using *dominance rules* or *elimination criteria* (see Emmons 1969). For the $1||\Sigma T_j$ the following dominance rule exists: If there is a pair of jobs $j$ and $k$ such that $p_j \le p_k$ and $d_j \le d_k$, then there exists an optimal sequence with job $j$ appearing in the sequence before job $k$. This rule makes it possible to reduce the amount of computation since it is not necessary to evaluate any sequence where job $k$ appears before job $j$. (However, it still is not possible to find a polynomial time algorithm for this problem, since it is known to be NP-hard.)

### *4.2.1. Numerical Example*

Consider three jobs with processing times $p_1 = 6$, $p_2 = 10$, $p_3 = 8$ and due dates $d_1 = 13$, $d_2 = 8$, $d_3 = 15$.

First, consider all sequencing problems that consist of a single job. If $S$ consists of either job 1 or job 3, then $G(S) = 0$ because these jobs would be completed before their due dates. If $S$ consists of job 2, then

$$G(S) = 10 - 8 = 2$$

Next, consider all problems that consist of two jobs. There are three possible sets $S$, namely $\{1, 2\}$, $\{1, 3\}$ and $\{2, 3\}$. Each one of the three problems has to be evaluated twice (because each job has to be given the chance to be the last one of the set to be completed). If $S = \{1, 2\}$, then the last job of $S$ is completed at $p_1 + p_2 = 16$. If job 1 is the last one of the set to be completed, then $T_1 = 3$ and

$$G(S - \{1\}) = G(\{2\}) = 2$$

If job 2 is the last one to be completed, then $T_2 = 8$ and

$$G(S - \{2\}) = G(\{1\}) = 0$$

So

$$G(\{1, 2\}) = \min(2 + 3, 0 + 8) = 5$$

The optimal order is first job 2 and then job 1. In the same way it can be determined that $G(\{1, 3\}) = 0$ and that in this case job 1 should precede job 3. The computations for set $\{2,3\}$ yield $G(\{2, 3\}) = 5$ and that job 2 should precede job 3.

Now consider all problems that consist of three jobs. There is only one such set, namely $\{1, 2, 3\}$. This set has to be evaluated three times because any one of the three jobs may be the last one to finish. From the fact that $p_1 + p_2 + p_3 = 24$, it follows that if job 1 is the last one to finish, $T_1 = 11$; if job 2 is the last one, then $T_2 = 16$; if job 3 is the last one, then $T_3 = 9$. So

$$\begin{aligned} G(\{1, 2, 3\}) &= \min(G(\{1, 2\}) + T_3, G(\{2, 3\}) + T_1, G(\{1, 3\}) + T_2) \\ &= \min(5 + 9, 5 + 11, 0 + 16) = 14 \end{aligned}$$

From these computations it is concluded that the optimal order is 2,1,3 and that the minimum value of the objective function is 14.

Actually, from the elimination criteria it could have been established in advance that job 1 has to precede job 3. The use of this piece of information throughout the procedure would have reduced the number of computational steps significantly.

## 4.3. Branch and Bound

In the scheduling field, the branch and bound technique appears to be more widely used than dynamic programming. The technique is basically an enumeration process that attempts to eliminate, through a bounding process, as many sequences as possible from consideration. The bounding process is very problem specific. Here the technique is applied to the $Fm||C_{\max}$ problem (see Ignall and Schrage 1965).

The branching process may be described as follows: A tree is built with a number of nodes at each level of the tree. The top level, level 1, of the tree consists of a single node, called the root. At this level the sequence is completely unspecified, that is, no job has a position in the sequence yet. This root has $n$ branches that go down to the second level. The $n$ nodes at the second level are characterized by the first job in the sequence, that is, at each node one particular job is assigned to the first position in the sequence and the positions of the $n - 1$ remaining jobs are still unspecified. Each node at level 2 has $n - 1$ branches emanating to the third level, with each node at the third level characterized by the jobs in positions 1 and 2 of the sequence. At each subsequent level there will be more nodes with fewer branches emanating to the next level.

The bounding process attempts to develop at any given node a lower bound on the objective functions of all sequences that start with the jobs specified at this node. A node at level $k + 1$ has $k$ jobs specified, say jobs $j_1, j_2, \ldots, j_k$; of the remaining $n - k$ jobs the positions still must be determined. Let $S$ denote these $n - k$ jobs and let $C_{i,j_k}$ denote the time job $j_k$ departs machine $i$ in the $Fm||C_{\max}$ example. In order to find one lower bound for all sequences which start with $j_1, j_2, \ldots, j_k$, observe that the first machine of the flow shop is always processing a job until the last job leaves the first machine. At best this last job goes through the remaining machines, after leaving the first machine, without having to wait for processing at any one of the remaining machines; thus,

$$C_{\max} \geq \sum_{j=1}^{n} p_{1j} + \min_{j \in S} \left( \sum_{i=2}^{m} p_{ij} \right) = LB_1$$

The first term of the lower bound represents the time the last job is completed on machine 1; the second term is the minimum time required for the last job to go through the remaining $m - 1$ machines. A second lower bound can be obtained by assuming that the second machine is continuously busy after $C_{2,j_k}$ and that the last job to leave the second machine goes through the remaining machines without having to wait for processing at any one of these machines. Thus,

$$C_{\max} \geq C_{2,j_k} + \sum_{j \in S} p_{2j} + \min_{j \in S} \left( \sum_{i=3}^{m} p_{ij} \right) = LB_2$$

In this way it is possible to obtain $m$ lower bounds. The last bound is

$$C_{\max} \geq C_{m,j_k} + \sum_{j \in S} p_{mj} = LB_m$$

The lower bound to use is the maximum of these $m$ lower bounds.

The search for the optimal sequence now goes as follows. One starts out with an initial sequence, which has to be chosen in a somewhat intelligent way (e.g., through a heuristic) and computes its $C_{\max}$. The better the initial sequence, the faster the branch and bound technique works. The procedure requires branching down the tree until a node is hit with a lower bound that is higher than the $C_{\max}$ of the best sequence found so far. This node is ‘‘then fathomed’’ and its offspring need not be considered. The search is then resumed from another node that appears to be promising (possibly because of a low lower bound at a node somewhat close to the root). When one finds a sequence with a $C_{\max}$ that is lower than the existing sequence one already has, one retains the new sequence and discards the other. The search is terminated when all nodes of the tree have been considered explicitly or implicitly (i.e., through fathoming).

### *4.3.1. Numerical Example*

Consider three machines and three jobs. The processing times of job 1 are $p_{11} = 8$, $p_{21} = 4$, $p_{31} = 5$, of job 2, $p_{12} = 1$, $p_{22} = 6$, $p_{32} = 3$ and of job 3, $p_{13} = 1$, $p_{23} = 9$, $p_{33} = 5$. An initial solution has to be obtained in order to have a bound to start out with. One can argue that it makes sense to let job 3 go first because it has a very small processing time on machine 1 and a very long processing time on machine 2. Choose sequence 3,2,1 as the initial sequence. Computing its makespan yields 25. This value is used as the initial bound. There are three nodes at level 2. The first node represents all sequences that start with job 1 in the first position (i.e., node (1,.,.) in Figure 3). Computing the bounds that are associated with this node yields:

$$LB_1 = 8 + 1 + 1 + \min(6 + 3, 9 + 5) = 19$$

$$LB_2 = 8 + 4 + 6 + 9 + \min(3,5) = 30$$

$$LB_3 = 8 + 4 + 5 + 3 + 5 = 25$$

So the lower bound at node (1,.,.) is 30. Because this lower bound is higher than the makespan of our initial sequence, it does not make any sense to evaluate the offspring of this node. Node (1,.,.) is thus fathomed. Evaluating node (2,.,.) and computing in a similar way a lower bound yields a value of 25. Because a sequence with a makespan of 25 is already known it is not necessary to evaluate the offspring of this node either; node (2,.,.) is therefore fathomed as well. Evaluating node (3,.,.) and computing a lower bound for this node yields a value of 23. Because this bound is lower than the makespan of the current sequence, the offspring of this node has to be evaluated; there may be a sequence with job 3 in the first position that has a makespan that is lower than the makespan of the current sequence. Evaluation of level 3, which turns out to be the last level, reveals that there are only two nodes at this level, namely nodes (3,1,2) and (3,2,1). Computing the makespan of sequence 3,1,2 yields a makespan of 23. This sequence is therefore optimal.

## 4.4. Decomposition Heuristics

In this section and the next, empirical techniques are presented that have proven to be useful in practice. These empirical techniques, or at least their underlying ideas, have been used in many existing industrial scheduling systems.

The technique discussed in this section has been designed for the $Jm||C_{\max}$ problem, which has received an enormous amount of attention in the literature. One of the most successful heuristic procedures developed for this problem is the shifting bottleneck heuristic (see Adams et al. 1988). A description of the procedure requires an alternative formulation of the problem.

Consider a directed graph $G = (N, A, B)$. The nodes $N$ correspond to all the operations to be done on the $n$ jobs. The (solid) arcs $A$ represent the precedence relationships between the various operations of the jobs. The disjunctive (broken) arcs $B$ form $m$ cliques of double arcs, one clique for each machine; the operations that are connected to one another in a clique have to be done on the same machine. *All* arcs, including the disjunctive ones, emanating from a node have as length the processing time of the operation at that node. In addition, there is a source and a sink, which are dummy nodes. The source (sink) has arcs emanating to (coming from) all the first (last) operations of the jobs. The arcs emanating from the source have length zero (see Figure 4).

A feasible schedule corresponds to a *selection* of one disjunctive arc from every pair in such a way that the resulting directed graph is acyclic. This implies, then, that each selection from a clique has to be acyclic. Such a selection, thus, determines the sequence in which the operations are to be performed on that machine. The length of such a schedule is, then, determined by the longest path

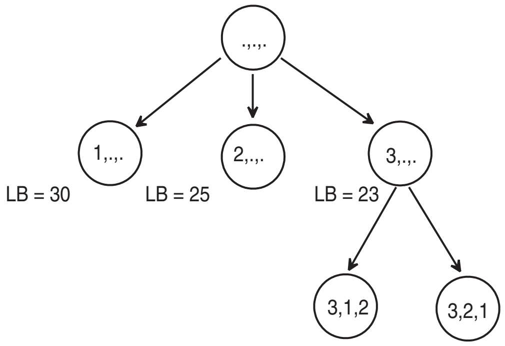


**Figure 3** Branch and Bound Tree.

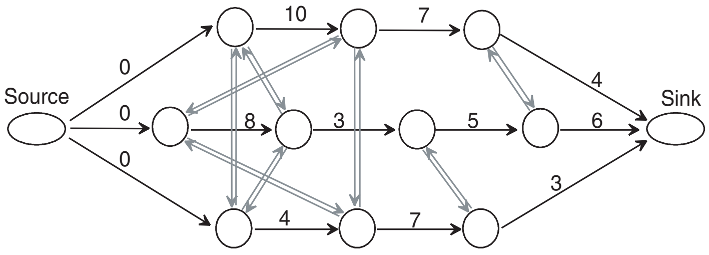


**Figure 4** Disjunctive Graph of Job Shop with Three Jobs and Four Machines.

(i.e., the *critical path*) from source to sink. The problem is to find the selections of disjunctive arcs which minimize the length of the longest path. The shifting bottleneck procedure now works as follows.

Let $M$ denote the set of all $m$ machines. Assume that for a subset of the machines, say $M_0$, the selections of disjunctive arcs have been determined. That is, job sequences on these machines have been determined. An additional machine has to be added to this subset and the sequence in which the operations are to be processed on this machine needs to be specified. To determine which machine should be the next one to be included in $M_0$, an attempt is made to determine which machine (among the ones still to be scheduled) causes in one sense or another the severest problems. In order to do this, the original directed graph is modified by deleting all disjunctive arcs of the machines still to be scheduled (i.e., machines in the set $M - M_0$) and keeping only the relevant disjunctive arcs of the machines already in $M_0$ (one from every pair). Call this graph $G'$. Deleting all disjunctive arcs of a specific machine implies that the associated operations, which originally were supposed to be done on the machine one after another, now can be done in parallel at any point in time (as if each one of these operations has its own private machine). The graph $G'$ has one or more critical paths as well as a makespan associated with it. Call this makespan $C_{\max}(M_0)$.

Assume now that each operation $j$ to be scheduled on machine $i$, $i \in M - M_0$, has to be processed in a time window of which the release dates and due dates are determined by the critical (longest) paths in $G'$. Consider each one of the machines in $M - M_0$ as a separate $1|r_j|L_{\max}$ problem where the maximum lateness has to be minimized (actually, the $1|r_j|L_{\max}$ problem is NP-hard, but algorithms have been developed for this problem that perform reasonably well). After these single machine problems are solved, it has to be determined which one of these single machine problems has the *largest* maximum lateness. This machine, in a sense, is the "bottleneck" among the remaining machines still to be scheduled and therefore the one to be added next to $M_0$. Label this machine $k$, call its maximum lateness $L_{\max}(k)$, and schedule it according to the optimal solution obtained. If the corresponding disjunctive arcs, which specify the sequence of operations on machine $k$, are inserted in graph $G'$, then the makespan increases by at least $L_{\max}(k)$, that is,

$$C_{\max}(M_0 \cup k) \geq C_{\max}(M_0) + L_{\max}(k).$$

Before the procedure is repeated and which machine to schedule next is decided, an additional step of resequencing each one of the machines in $M_0$ needs to be done. That is, a machine is taken out of the set $M_0 \cup k$, say machine $l$. A graph $G''$ has to be constructed in the same way as graph $G'$ was constructed including now the disjunctive arcs which specify the sequence of operations on machine $k$ and excluding the disjunctive arcs associated with machine $l$; machine $l$ has to be resequenced by solving the corresponding $1|r_j|L_{\max}$ problem with the release and due dates determined by the critical paths in graph $G''$. After this resequencing is done for each one of the machines in the original set $M_0$, the entire procedure is repeated in order to add another machine to the current set $M_0 \cup k$.

The structure of this heuristic shows the relationship between the bottleneck concept and the more combinatorial concepts such as the critical (longest) path and the maximum lateness. A critical path indicates the location and timing of a bottleneck. The maximum lateness indicates the amount with which the makespan increases if a machine is added to the set of machines already scheduled.

Extensive numerical research has shown that this heuristic is extremely effective. Applied on a particular test problem with 10 machines and 10 jobs, which had remained unsolved for more than 20 years, the heuristic obtained a very good solution after only a couple of minutes of CPU time. This solution turned out to be optimal after a branch and bound approach, applied to the problem, obtained the same result and verified its optimality. The branch and bound approach, in contrast to the heuristic, needed many hours of CPU time. The disadvantage of the heuristic is, of course, that whether an optimal solution actually has been reached can never be known for sure.

## 4.5. Local Search

Simulated annealing and taboo search are two techniques that can be viewed as generalizations of the iterative improvement approach to combinatorial optimization problems.

Simulated annealing originated in a different field: it was first developed as a simulation model for describing a physical annealing process for condensed matter. The application of simulated annealing to scheduling requires a certain amount of structure (see Matsuo et al. 1989). At stage $k$ of the process, there is a solution for the scheduling problem; for a single-machine problem this solution is merely a given sequence (permutation) of the jobs, say $\sigma_p$. Let $G(\sigma_p)$ denote the value of the objective function using this solution. For this solution a *neighborhood* can be defined. If the solution is just a permutation of the $n$ jobs, then the neighborhood could be defined as all permutations that can be obtained by interchanging a pair of adjacent jobs (which implies that the neighborhood then consists of $n - 1$ different sequences). Clearly, the structure of the neighborhood can be made more complicated and is a design issue. In order to be allowed to move from solution $\sigma_p$ to solution $\sigma_q$, which is an element of the neighborhood of solution $\sigma_p$ an acceptance probability

$$P_{pq}(k) = \min\left\{1,\exp\left[-\frac{G(\sigma_q) - G(\sigma_p)}{\beta_k}\right]\right\}$$

is defined, where $k$ is the so-called stage of the search and $\beta_1 \geq \beta_2 \ldots$ The stage is a level in which the same acceptance probability is used, and $\beta_k$ is a control parameter. This $\beta_k$ tends to zero as $k$ increases, which implies that the acceptance probability for a move to a worse solution is lower at a later stage in the process. From the definition of the acceptance probability, it also follows that the worse a neighbouring solution is, the lower the acceptance probability is.

The simulated annealing procedure now works as follows. At each stage a series of neighborhood searches is done. A search can be done in a random way or in a organized (possibly sequential) way. A neighbor is compared with the "seed" (current) solution. When the value of the objective function of the neighbor is less than the value of the objective of the seed, the neighbor is automatically accepted and becomes the new seed. If the value of the objective of the neighbor is higher than the one of the seed, the neighbor is accepted as the new seed with a probability that is determined by the acceptance probability. However, the best solution obtained so far is always being kept in memory. In practice, several stopping criteria are used for this procedure. One way is to let the procedure run for a given (prespecified) number of iterations. Another is to let the procedure run until for a given number of iterations no improvement has been obtained.

Taboo search is in many ways similar to simulated annealing. The procedure moves again from one solution to another, with the next solution being possibly worse than the preceding solution. For each solution (or sequence) a neighborhood is defined, possibly in exactly the same way as it is defined for simulated annealing. The reason for allowing a solution to be worse than the previous one is to give the procedure the opportunity to move away from a local minimum and have a chance to find a lower minimum. The mechanism that guides the moves is different from the one in simulated annealing (see Glover 1990). At any stage of the process a so-called taboo list is being kept. This list contains the moves to the neighboring solutions the procedure is *not* allowed to make. This list has a fixed number of entries (this number usually lies between 5 and 9). Every time a move is made, the reverse move is put at the top of the taboo list; all other entries are pushed one position down and the bottom entry is deleted. The reason for putting the reverse move in the list is to avoid a move back to a local minimum that has been visited before. The search for a neighbor to which the procedure is allowed to move to is a design issue. This can, just as in simulated annealing, be done in a random way or in an organized (sequential) way.

The use of simulated annealing and taboo search has its advantages and disadvantages. One advantage is that it can be applied to a problem without having to know much about the structural properties of the problem. It can be coded very easily and it gives solutions that are usually fairly good. However, the amount of computation time needed to obtain such a solution tends to be relatively long in comparison with more rigorous problem-specific approaches.

Simulated annealing as well as taboo search are often used in the following manner. First an attempt is made to find a reasonably good initial solution for a problem via a heuristic. After this has been done, either a simulated annealing or a taboo search procedure is used as a postprocessor in order to search for an even better solution. The postprocessor is then run for a given amount of time.

### 4.6. Multiple Objectives

Little of the theoretical research done in the past has dealt with multiple-objective models. There are, however, several approaches for dealing with such problems. We illustrate one approach here through an example (see McCormick and Pinedo 1995).

Consider the problem $Pm|prmp|\alpha_1\Sigma C_j + \alpha_2 C_{\max}$; that is, there are $m$ identical machines in parallel with preemptions allowed and as the objective the minimization of a weighted sum of makespan and flow time. It can be shown fairly easily that the shortest processing time first (SPT) rule minimizes $\Sigma C_j$ on parallel machines, while the preemptive longest remaining processing time first (preemptive LPT) rule minimizes $C_{\max}$. Because these two rules are quite different, it is not immediately obvious what type of rule minimizes a mixture of these two objectives.

The problem can be analyzed by transforming one of the objectives into a constraint; that is, consider the problem where the flow time has to be minimized subject to the makespan being smaller than or equal to a given deadline $d$. This problem turns out to have a fairly simple solution. Without loss of generality, it may be assumed that $p_1 \leq p_2 \leq \ldots \leq p_n$. The scheduler schedules the jobs according to SPT until time $d - p_n$. At this point in time this largest job *must* be started on one of the machines, preempting the largest job among the ones being processed at that moment. The scheduler then continues using SPT until the second-largest job must be started, that is, at $d - p_{n-1}$. At this point in time the largest job being processed, not including job $p_n$, is preempted by job $n - 1$, and so on. It is easy to compute the value of the objective function, given $C_{\max} = d$. Through a parametric analysis on $d$, one can determine the minimum flow time as a function of the makespan. This function is decreasing convex and piecewise linear with a number of breakpoints (see Figure 5).

The values of $\alpha_1$ and $\alpha_2$ determine at which breakpoint the optimal solution lies.

In general, when there are a number of objectives to be minimized, the following heuristic approach can be used. Select as the first (principal) objective to minimize one that is important (that is, has a large weight) as well as sensitive to the schedule. While one optimizes this objective, one continuously keeps the other objective functions (which are of lesser importance) in mind. For example, whenever a tie needs to be broken, it is broken in a way that is beneficial for a secondary objective. After having completed the optimization procedure for the first objective, one proceeds with considering a second objective, and so on.

## 5. SCHEDULING PROBLEMS IN PRACTICE

This section focuses on a number of application areas where dispatching and scheduling techniques are of importance.

### 5.1. Real-Life Scheduling Problems vs. Theoretical Models

Real-life scheduling problems usually are very different from the mathematical models studied by researchers in academia and industrial research centers. It is difficult to categorize all differences between the real problems and the theoretical models, as each real-life scheduling problem has its own idiosyncrasies. Nevertheless, a number of these differences do stand out and are worth mentioning.

Theoretical models usually assume that there are $n$ jobs to be scheduled and that after these $n$ jobs are scheduled the problem is solved. In real life there may be at any point in time $n$ jobs in the system, but every day (week or month) new jobs are added. Scheduling the current $n$ jobs has to be

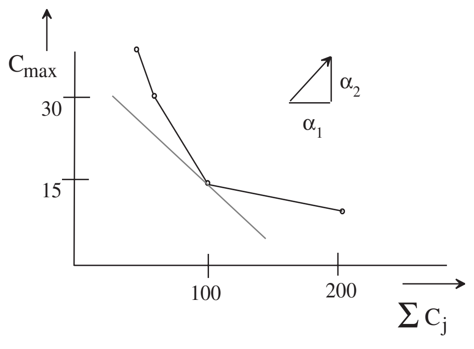


**Figure 5** Trade-off Curve between Makespan and Flow Time.

done without having a perfect knowledge of what will happen in the near future. However, some provisions have to be made in order to be prepared for the unexpected. The dynamic nature of scheduling problem may therefore require, for example, slack times to be built into the schedule.

The models usually do not emphasize the *resequencing* problem. In real life the following problem often occurs. There exists a schedule that was generated based on certain assumptions; now an (unexpected) event has occurred that requires either major or minor modifications in the existing schedule. The rescheduling process, which is sometimes referred to as reactive scheduling, may have to satisfy certain constraints. For example, one may wish to keep the changes in the existing schedule at a minimum, even if an optimal schedule cannot be achieved this way.

The models usually do not consider *preferences.* In a model, a job either can or cannot be processed on a given machine. In reality, it often occurs that a job *can* be scheduled on a given machine but that there is a *preference* (for one reason or another) not to schedule it on the machine in question; scheduling it on the machine in question would only be done in case of an emergency.

Most of the theoretical research has been focused on models with a single objective. In real life there are, of course, a large number of objectives to deal with. Not only are there many objectives, but their respective weights may vary over time and may even depend on the particular person in charge.

In spite of the many differences between real-life scheduling problems and the mathematical models discussed in the previous sections, the general consensus is that the theoretical research done in the past has not been a complete waste of time. It has provided valuable insights into many scheduling problems, and these insights have proven to be useful in the development of a large number of scheduling systems.

## 5.2. Scheduling in the Packaging Industry

Consider a factory that produces paper bags for cement, dog food, charcoal, and so on. A scheduling system for such a factory has to be based on a flexible flow-shop model (see Adler et al. 1993). That is, there are a number of stages in series (for example, printing, glueing, and sewing) and at each stage there are a number of machines in parallel. The machines at any given stage may not be identical. Some machines may be more modern than others and may run at a higher speed or may be able to handle more complicated jobs than other machines. The main objectives are to meet the committed shipping dates as much as possible while minimizing the setup times on the machines. The algorithmic procedures adopted in such a system have to follow a number of steps.

First, the procedure goes through a bottleneck-identification process. At least one of the stages is a bottleneck. The identification of the bottleneck(s) can be done manually (specified by the user) or computed using all machine and job data. Then the procedure computes time windows during which jobs have to be processed at the bottleneck stage. The earliest time a job is allowed to start at a bottleneck stage depends on its current status upstream, while the latest time a job is allowed to be finished depends on its committed shipping date as well as on the amount of processing that is needed downstream. The current status upstream of a job may be either the stage the job currently is being processed at or the time the raw material (paper board) is expected to arrive at the facility and the job can be started at the first stage.

After this has been done, the procedure computes the various machine capacities at the bottleneck stage. For each machine it is computed how the capacity compares with the amount of processing that *has to* be done on it (that is, jobs that cannot be processed on any other machine) and with the amount of processing that *can* be done on it (that is, jobs that can be processed on other machines as well). This computation is done for various time periods (one week ahead, two weeks ahead, etc.).

Now scheduling of machines at the bottleneck stage can be done based on the information compiled in the previous steps. This schedule attempts to process all jobs within their respective windows while minimizing setup times. The procedure used here is a generalization of the ATC rule, which includes setup times. After the machines at the bottleneck stage have been scheduled, the machines at all other stages can be scheduled. The order in which the jobs go through the machines upstream and downstream of the bottleneck stage is somewhat similar to the order in which the jobs go through the bottleneck stage, with minor adjustments to improve on setup times or accommodate for machine preferences.

## 5.3. Scheduling in the Semiconductor Industry

In recent years a great amount of work has been done on wafer fabrication scheduling (see Hadavi and Voigt 1987). The reason for this activity is that equipment as well as products are extremely expensive. There is a great deal of randomness in the process; machines break down often and processing times are random. The machine environment can be considered as a job shop or as a flowshop with recirculation.

A system in such an environment has to create high-level production plans as well as more and more detailed scheduling plans. The timing of the generation of these detailed schedules depends on the urgency or the importance of the jobs in question.

While a large number of existing scheduling systems rely on good heuristics for solving the actual current scheduling problems (that is, *reacting* to real-time problems on the shop floor), some systems may take the more rational approach of *preventive* scheduling (that is, scheduling to avoid as many future problems as possible while at the same time optimizing the performance). In order to do so, such a system has to rely on a job-release mechanism that has a global view of the factory. Such a job-release mechanism attempts to achieve a maximal level of machine utilization and a minimal number of job delays. It may attempt to achieve this by monitoring a job parameter such as the *continuity index,* which measures how and when the job will be processed if released as proposed. The continuity index of a job is affected by bottleneck work centers, inventory shortages, tool availability, and so on. Based on these estimates, job releases are planned as much as possible to alleviate the prospective bottlenecks. This tends to minimize cycle times, which is one of the main objectives in VLSI development lines.

The sequencer and shop-floor control module of a scheduling system may track the work in progress as well as the status of the machines. It performs reactive scheduling functions based on this informtion. Such a module may be based on the axiom of *locality*, which implies that if an unexpected event occurs, an effort is made to limit, as much as possible, the number of changes in the existing schedules when correcting the problem.

The ReDS system developed by Siemens AG uses the real-time shop-floor data for another purpose as well (besides rescheduling). It uses the data in the form of a long-term feedback in order to make adjustments in the heuristics employed. This learning mechanism enables a scheduling system to mold itself over time into the shape of the factory in which it is operating.

## 5.4. Scheduling in the Automotive Industry

In the automotive industry, production scheduling ranges from job-shop scheduling in parts/components assembly to detailed job sequencing in automobile assembly. In automobile assembly, job sequencing and assembly line balancing are two concepts that are very closely related (see Burns and Daganzo 1987; Yano and Bolat 1990). This is largely due to two characteristics. First, most automobile assembly lines are mixed-model assembly lines with different processing time requirements for a large number of different models. In this case, different models of jobs may at times differ only in the options they carry and at other times may constitute completely different bodies. Secondly, most automobile assembly lines are paced, that is, jobs are processed at a constant rate that is determined by different model mixes and processing time requirements.

After the assembly line rate is determined, the operations with longer processing times are generally performed in sufficiently long sections of the line. This is possible because a large part of the assembly process is manually operated. However, job sequencing is now complicated because of the fact that the different operations performed on the different models require that similar jobs be adequately spaced in the sequence such that a proper balance of workload is achieved. Poor sequencing could result in reduced production and possibly quality problems. For example, suppose 10% of the jobs (cars) need a sunroof. Suppose that for a typical operation, to be done on *every* job, the processing time is $p$. The processing time for installing a sunroof is $3p$ since the amount of time it takes to install a sunroof is significant. The sunroof operation is designed to hold five jobs, with each job spending a total time of $6p$ in the operation (i.e., jobs are processed at a rate of $5/6p$). However, this implies that if there are three or more consecutive jobs requiring a sunroof, the operation is overloaded. An attempt is therefore made to place jobs with sunroofs as far apart as possible (i.e., approximately every tenth job) because otherwise the operation might be delayed and quality problems might occur if the workers do not have sufficient time to perform the operation properly.

Thus, the assembly line has to be sequenced in a way that more or less balances the workload at all operations, especially at those that in one form or another are critical. The operations most likely to be critical are those with a workload that tends to vary significantly from job to job.

Job sequencing is further complicated by the requirements that certain jobs be grouped together. For example, the paint process requires that each time the color of a job changes, the previous paint color must be purged from the spray gun to avoid paint overspray. Minimizing the paint purges can help reduce the paint cost, and therefore it is desirable to group jobs with the same color together in the job sequence. Also, to coordinate a better shipping schedule for finished automobiles, jobs shipped to the same destination should be grouped roughly in the same time interval.

Several large car manufacturers have developed integrated scheduling systems that take all the above requirements into account. Several systems are being used on a daily basis.

## 5.5. Scheduling in the Aviation Industry

In the aviation industry, many scheduling problems arise, such as crew scheduling, maintenance scheduling, and so on. In this section, a system is described that assigns airplanes to gates at an airport (see Brazile and Swigger 1988). Such scheduling systems have been developed to assign airplanes to gates at various airports in the United States, Europe, and Japan.

The problem can be viewed as a scheduling problem with machines in parallel. The gates represent the machines and the airplanes represent the jobs. The jobs have release dates that are the arrival times of the planes. These release dates are subject to random factors, such as weather and equipment failure. Any given job needs to be processed on one of the machines; the processing time is equivalent to the turnaround time of the plane. Some jobs can only be processed on a specific subset of the machines; for example, large planes can only go to specific gates while small planes may go to any one of the gates. The objective is to find a feasible assignment of jobs to machines.

Some of the systems are knowledge-based systems that attempt to find a feasible schedule through constraint satisfaction. They contain rules that guide the search and are capable of producing the trail of the rules used in reaching the conclusion. These systems may operate in one or two modes. In one mode, the static mode, it produces a daily schedule, which is followed only if everything goes as planned (which, of course, hardly ever happens). In the second mode, the dynamic mode, the system reassigns the gates in real time as information about changes in arrival and departure times becomes available.

Various different types of constraints with regard to the gate-assignment process can be formulated. There are "hard" constraints; for example, certain gates simply cannot accomodate 747s. There are "soft" constraints, which do allow exceptions; for example, domestic flights do not necessarily always have to arrive at and depart from domestic gates. There are "convenience" constraints; for example, planes that are scheduled to remain many hours at the terminal preferably should not be moved from one gate to another.

To arrive at an assignment, a system may search its way through two types of rules, permissive rules and conflict rules. Permissive rules determine whether it is appropiate to consider a particular gate for a particular flight. For example, when a plane is not continuing the same day, using a remote gate is acceptable. Conflict rules basically embody in the program the hard constraints mentioned before.

Systems may use a number of priority rules in their assignment of flights to gates. They first assign flights and gates that are the most constrained and the hardest to assign. Because only a small number of gates are capable of handling 747s, the wide-bodied aircraft are assigned first. Certain gates are so close to one another that planes cannot taxi in but have to be towed in. These gates have the lowest priority in the assignment.

Gate assignments are made when no rules are violated. If a feasible assignment is obtained in one pass, the process terminates. If the system does not find a feasible assignment in one pass, it automatically deactivates certain optional rules and tries again.

A system's second mode (the reassignment mode) operates as follows. It receives the most recent data concerning the current status of arriving and departing flights from the airline's database. When deviations are larger than given treshold values, the system checks whether the changes are creating conflicts. If there is a conflict, the system attempts to create a new schedule with a minimum number of changes. At times it may invoke the help of the scheduler.

## 6. DEVELOPMENT OF SCHEDULING SYSTEMS

### 6.1. General Structure of Scheduling Systems

During the last decade, numerous computer-based scheduling systems have been developed, many of which are currently controlling the scheduling operations in a variety of industries. For a number of reasons, the implementation of such systems usually turns out to be at least as difficult as the actual development. The development of these systems has taken place at R&D centers of industrial corporations as well as at universities (see Kanet and Adelsberger 1987; Adelsberger and Kanet 1989). Computer-based scheduling systems often consist of three modules:

1. A database management module
2. A schedule-generation module
3. A user interface module

See Figure 6.

All three parts play a crucial role in the functionality of the system. In practice, a significant amount of effort is usually required to make a factory's database suitable for input to the system. Making the database accurate, consistent, and complete often involves the design of a series of tests the data must pass before they can be used. This module may also have capabilities of manipulating the data, performing various forms of statistical analysis, and enabling the scheduler to see data in the form of bar charts or pie charts. The schedule-generation module involves the formulation of a suitable model, the formulation of objective functions and/or constraints, and (possibly) the development of the algorithms. The user interface module is very important, especially with regard to the implementation process. Without a good user interface, there is a good chance that, regardless of its

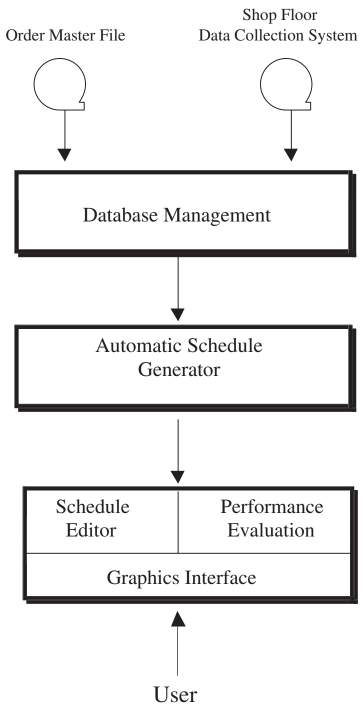


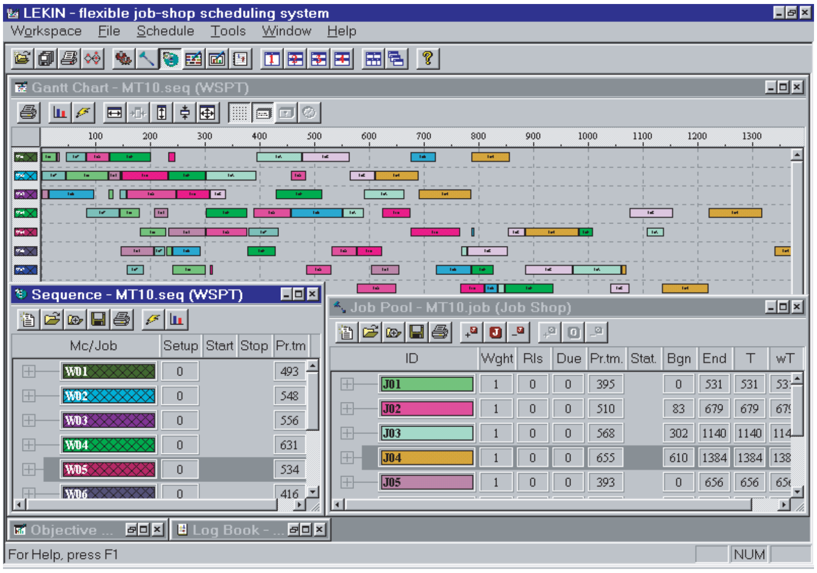
**Figure 6** Configuration of a Scheduling System.

scheduling capabilities, the system will never be used. This user interface often takes the form of an electronic Gantt chart with tables and graphs that enable the scheduler to edit the schedule generated by the system and take last-minute information into account (see Figure 7). When the scheduler edits the schedule generated by the system, he or she is usually able to follow the impact of his or her changes on the various measures of performance as well as compare several schedules with one another and perform an extensive what-if analysis.

## 6.2. Schedule Generation

There are several schools of thought with regard to schedule generation. Two of these deserve further discussion. One, which is predominantly used by industrial engineers and operations researchers, could be called the *algorithmic* approach. The other, which is often used by computer scientists and artificial intelligence experts, is usually called the *knowledge-based* approach (see Kanet and Adelsberger 1987). Recently, these two approaches have started to converge towards one another and the differences have become more blurry. Some hybrid systems developed in the recent past use a knowledge base as well as fairly sophisticated heuristics.

The first approach usually requires a mathematical formulation of the problem, which includes objectives and constraints. The algorithm could be based on any one of the techniques or combination of techniques presented in Sections 3 and 4. The ''goodness'' of the solution is based on the values of the objectives and performance criteria under the given schedule. This form of schedule generation often may consist of three segments. In the first segment a certain amount of *preprocessing* is done. In this segment the problem instance is analyzed and a number of statistics are compiled, such as the average processing time, the maximum processing time, the due date tightness. The second segment consists of the actual algorithms and heuristics. The structure of the algorithm or heuristic may depend on the statistics compiled in the first segment (for example, in the way the look-ahead parameter $K$ in the ATC rule may depend on the due date tightness and due date range factors). The third segment may contain a *postprocessor*. The solution that comes out of the second segment is fed into a procedure such as simulated annealing or taboo search in order to see whether any improvements can be obtained. This type of schedule generation is usually coded in a procedural language such as Fortran, Pascal, or C.


LEKIN - flexible job-shop scheduling system
Workspace File Schedule Tools Window Help
Gantt Chart - MT10.seq (WSPT)
100 200 300 400 500 600 700 800 900 1000 1100 1200 1300
Sequence - MT10.seq (WSPT)
Job Pool - MT10.job (Job Shop)
Objective ...
Log Book - ...
For Help, press F1
NUM

| Mc/Job | Setup | Start | Stop | Pr.tm |
|---|---|---|---|---|
| W01 | 0 | | | 493 |
| W02 | 0 | | | 548 |
| W03 | 0 | | | 556 |
| W04 | 0 | | | 631 |
| W05 | 0 | | | 534 |
| W06 | 0 | | | 416 |

| ID | Wght | Rls | Due | Pr.tm. | Stat. | Bgn | End | T | wT |
|---|---|---|---|---|---|---|---|---|---|
| J01 | 1 | 0 | 0 | 395 | | 0 | 531 | 531 | 53 |
| J02 | 1 | 0 | 0 | 510 | | 83 | 679 | 679 | 67 |
| J03 | 1 | 0 | 0 | 568 | | 302 | 1140 | 1140 | 114 |
| J04 | 1 | 0 | 0 | 655 | | 610 | 1384 | 1384 | 138 |
| J05 | 1 | 0 | 0 | 393 | | 0 | 656 | 656 | 65 |

**Figure 7** The Gantt Chart Interface of the Lekin System.

The second approach is different from the first in various respects. This approach is often more concerned with underlying problem structures that cannot easily be described in an analytical format. In order to incorporate the scheduler's knowledge into the system, so-called rules and objects are used. This approach is used often when it is only necessary to find a *feasible* solution given the many constraints or rules; however, because some schedules are ranked "more preferable" than others, heuristics are used at times in order to obtain a "preferred" schedule. Through an *inference engine* the approach attempts to find sequences that do not violate prescribed rules and satisfy stated preferences as much as possible. Whenever a satisfactory solution does not appear to exist or when the scheduler judges it to difficult to find, the scheduler may reformulate the problem through a relaxation of the constraints. The relaxation of constraints may actually be done automatically by the system itself. The programming style used in the development of such systems is usually different from the ones used under the first approach; systems are usually coded in languages that have so-called object oriented extensions, such as LISP and C++. These languages emphasize user-defined functions, which promote a modular programming style.

Both approaches have their advantages and disadvantages. The algorithmic approach clearly has an edge if (1) the problem allows for a crisp mathematical formulation, (2) the number of jobs involved is large, (3) the amount of randomness in the environment is minimal, and (4) some form of optimization has to be done frequently and in real time (it is very common that schedulers have neither the patience nor the time to wait more than 30 seconds for a schedule to be specified). One disadvantage of the algorithmic approach is that if the scheduling environment changes (for example, certain preferences on assignments of jobs to machines), the reprogramming effort may be substantial. The knowledge-based approach may have an edge if only a *feasible* schedule is required. Some system developers believe that changes in the scheduling environment or rules can be incorporated more easily in a knowledge-based system than in a system that is based on the algorithmic approach. Others, however, believe that the effort required to modify any system is mainly a function of how well the code is organized and written; the effort required to modify does not depend that much on the approach used. One disadvantage of the knowledge-based approach is that obtaining a reasonable schedule may take a substantial amount of computer time.

## 6.3. Software Development and Implementation

The last three decades have seen the design and implementation of many scheduling systems. Some of these systems were application specific and others were generic. Some were developed for research and experimentation and others were commercial.

A number of scheduling systems have been designed and developed in academia over the last three decades. Several universities developed research systems or educational systems that were often based on ideas and algorithms that were quite novel. An example of such a system is the Lekin system, which can be downloaded free of charge from the Web. Some of the academic systems have been handed over to industry and have led to the start-up of software companies.

The last two decades have witnessed the development of scores of commercial scheduling systems. There were a few major trends in the design and development of these commercial scheduling systems.

One trend started in the 1980s when a number of companies began to develop sequencing and scheduling software. Most of these companies tended to focus in the beginning only on sequencing and scheduling. They started out with the development of generic scheduling software that was designed to optimize flow lines or other types of machine environments. Some of these companies have grown significantly since their inception, such as ILOG, I2, and Manugistics.

These companies, whenever they landed a contract, had to customize their software to the specific applications. Because they realized that customization of their software customization is a way of life, they usually tried to keep their schedule generators as generic as possible. The optimization methodologies they adopted often included:

1. Shifting bottleneck procedures
2. Local search procedures
3. Mathematical programming procedures

These companies, which at the outset were focusing primarily on sequencing and scheduling, began to branch out in in the 1990s; they started to develop software for supply chain management. This diversification became necessary because clients typically had a preference for dealing with vendors that could provide a suite of software modules capable of optimizing the entire supply chain; clients did not like to have to deal with different vendors and face all kinds of integration problems.

A second major trend in the development of sequencing and scheduling software had its source in another corner of the software industry. This second trend started to take place in the beginning of the 1990s. Scheduling software started being developed by companies that at the outset specialized in ERP systems, such as SAP, Baan, J.D. Edwards, and PeopleSoft. These ERP systems basically are huge accounting systems that serve as a backbone for all the information requirements in a company. This backbone can then be used to feed information into all kinds of decision support systems, such as forecasting systems, inventory control, and sequencing and scheduling. The software vendors that specialized in ERP systems realized that it was necessary to branch out and develop decision support systems as well. A number of these companies either bought a scheduling software company (e.g., Baan bought Berclain), started their in-house scheduling software development (e.g., SAP), or established partnerships with scheduling software vendors.

Currently there are more than a hundred scheduling software vendors. Most of these are relatively small. The bigger players are I2, Cybertec, and Manugistics, all of them offering software for the entire supply chain. The main ERP vendors, such as SAP, Baan, PeopleSoft, and J.D. Edwards, all offer sequencing and scheduling packages. Some of their scheduling modules had been developed internally, whereas other modules were developed through acquisitions of smaller software companies specializing in scheduling. The algorithmic approaches differ considerably from company to company. Some companies specialize in local search procedures, while others specialize in mathematical programming techniques, and again others in decomposition techniques. Even the preferences for user interfaces may differ. However, the most popular user interface tends to be the Gantt chart.

## 7. CONCLUDING REMARKS

During the last two decades, with the advent of the personal computer in the factory, a large number of scheduling systems has been developed. Currently, many more scheduling systems are under development. This developmental process has made it clear that a large proportion of the theoretical research done during the decades past is of limited use in real-world applications. The system development that is currently going on in industry is fortunately encouraging theoretical researchers to tackle scheduling problems that are more relevant to the real world. At various universities in Europe, Japan, and North America, research is being focused on the development of algorithms as well as on the development of systems; significant efforts are being made in the integration of these developments (see McKay et al. 1989).

Even though during the last decade many companies have made large investments in the development and implementation of scheduling systems, not that many systems appear to be used used on a regular basis. Systems, after being implemented, often remain in use only for a limited time; after a while they are often, for one reason or another, ignored altogether.

In those situations where the systems are in use on a more or less permanent basis, the general feeling is that the operations do run more smoothly. A system in place usually does *not* reduce the time the scheduler spends on the scheduling process. However, a system usually does enable the scheduler to produce better schedules. Using an interactive schedule editor, the scheduler is able to compare different schedules and easily monitor the various performance measures. Actually, there are other reasons for smoother operations besides simply better schedules. A scheduling system imposes more "discipline" on the operations. There are compelling reasons now for keeping an accurate database. Schedules are printed out neatly and are visible on monitors. This apparently has an effect on people, encouraging them to actually even obey the schedules.

It would be interesting to know the reasons why some systems have never become implemented or are never used. In some cases databases are not sufficiently accurate. In other cases the way in which workers' productivity is measured is not in agreement with the performance criteria the system is based upon. User interfaces may not enable the scheduler to resequence quickly in the case of unexpected events. There may also be an absence of procedures that enable resequencing when the scheduler is absent (for example, if something unexpected happens during third shift). Finally, systems may not be given sufficient time to settle or stabilize in their environment (this may require many months, if not years).

Nevertheless, it appears that in the decade to come an even larger effort will be made in the development of such systems and that such systems will play an important role in computer-integrated manufacturing.

## REFERENCES

Adams, J., Balas, E., and Zawack, D. (1988), "The Shifting Bottleneck Procedure for Job Shop Scheduling," *Management Science,* Vol. 34, pp. 391–401.

Adelsberger, H. H., and Kanet, J. (1989), "The Leitstand—A New Tool in Computer-Aided Manufacturing Scheduling," Technical Report, College of Commerce and Industry, Clemson University, Clemson, SC.

Adler, L., Fraiman, N. M., Kobacker, E., Pinedo, M. L., Plotnicoff, J. C., and Wu, T. P. (1993), "BPSS: A Scheduling System for the Packaging Industry," *Operations Research,* Vol. 41, pp. 641–648.

Brazile, R. P., and Swigger, K. M. (1988), "GATES: An Airline Gate Assignment and Tracking Expert System," *IEEE Expert,* Vol. 3, No. 2, pp. 33–39.

Burns, L., and Daganzo, C. F. (1987), "Assembly Line Job Sequencing Principles," *International Journal of Production Research,* Vol. 25, pp. 71–99.

Chen, N.-F., and Liu, C. L. (1975), "On a Class of Scheduling Algorithms for Multiprocessors Computing Systems," in *Parallel Processing,* Lecture Notes in Computer Science 24, T. Y. Feng, Ed., Springer, Berlin, pp. 1–16.

Emmons, H. (1969), "One-Machine Sequencing to Minimize Certain Functions of Job Tardiness," *Operations Research,* Vol. 17, pp. 701–715.

Glover, F. (1990), "Tabu Search: A Tutorial," *Interfaces,* Vol. 20, Issue 4, pp. 74–94.

Graham, R. (1969), "Bounds on Multiprocessing Anomalies," *SIAM Journal of Applied Mathematics,* Vol. 17, pp. 263–269.

Hadavi, K., and Voigt, K. (1987), "An Integrated Planning and Scheduling Environment," in *Proceedings of AI in Manufacturing Conference* (Long Beach, CA).

Held, M., and Karp, R. M. (1962), "A Dynamic Programming Approach to Sequencing Problems," *Journal of SIAM,* Vol. 10, pp. 196–210.

Ignall, E., and Schrage, L. E. (1965), "Application of the Branch and Bound Technique to Some Flow-Shop Problems," *Operations Research,* Vol. 13, pp. 400–412.

Jackson, J. R. (1955), "Scheduling a Production Line to Minimize Maximum Tardiness," Research Report 43, Management Science Research Project, University of California, Los Angeles.

Kanet, J., and Adelsberger, H. H. (1987), "Expert Systems in Production Scheduling," *European Journal of Operational Research,* Vol. 29, pp. 51–59.

Kawaguchi, T., and Kyan, S. (1986), "Worst Case Bound of an LRF Schedule for the Mean Weighted Flow Time Problem," *SIAM Journal of Computing,* Vol. 15, pp. 1119–1129.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., and Shmoys, D. B. (1989), "Sequencing and Scheduling: Algorithms and Complexity," Report BS-R8909, Centre for Mathematics and Computer Science, Amsterdam.

Lee, Y.-H., Bhaskaran, K., and Pinedo, M. L. (1997), "A Heuristic to Minimize the Total Weighted Tardiness with Sequence Dependent Setups," *IIE Transactions,* Vol. 29, pp. 45–52.

Matsuo, H., Suh, C. J., and Sullivan, R. S. (1989), "A Controlled Search Simulated Annealing Method for the Single Machine Weighted Tardiness Problem," *Annals of Operations Research,* Vol. 20, pp. 85–108.

McCormick, S. T., and Pinedo, M. L. (1995), "Scheduling *n* Independent Jobs on *m* Uniform Machines with Both Flow Time and Makespan Objectives: A Parametric Analysis," *ORSA Journal of Computing,* Vol. 7, pp. 63–77.

McKay, K. N., Buzacott, J. A., and Safayeni, F. (1989), "The Schedulers Information System—What is Going on? Insights for Automated Environments," in *Proceedings of the II-COM 1989 Conference* (Madrid).

Rosenkrantz, D. J., Stearns, R. E., and Lewis, P. M. (1977), "Approximate Algorithms for the Travelling Salesman Problem," *SIAM Journal of Computing,* Vol. 6, pp. 543–558.

Smith, W. E. (1956), "Various Optimizers for Single Stage Production," *Naval Research Logistics Quarterly,* Vol. 3, pp. 59–66.

Vepsalainen, A., and Morton, T. (1987), "Priority Rules for Job Shops with Weighted Tardiness Costs," *Management Science,* Vol. 33, pp. 1035–1047.

Yano, C. A., and Bolat, A. (1990), "Survey, Development and Applications of Algorithms for Sequencing Paced Assembly Lines," *Journal of Manufacturing and Operations Management,* Vol. 3, pp. 172–198.

## ADDITIONAL READING

Baker, K. R., *Introduction to Sequencing and Scheduling,* John Wiley & Sons, New York, 1974.

Brucker, P., *Scheduling Algorithms,* 2nd Ed. Springer, Berlin, 1998.

Conway, R. W., Maxwell, W. L., and Miller, R. W., *Theory of Scheduling,* Addison-Wesley, Reading, MA, 1967.

Dempster, M. A. H., Lenstra, J. K., and Rinnooy Kan, A. H. G., Eds., *Deterministic and Stochastic Scheduling,* Reidel, Dordrecht, 1982.

French, S., *Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop,* Horwood, Chichester, 1982.

Pinedo, M., *Scheduling: Theory, Algorithms and Systems,* Prentice Hall, Englewood Cliffs, NJ, 1995.

Pinedo, M., and Chao, X., *Operations Scheduling with Applications in Manufacturing and Services,* Irwin/McGraw-Hill, Boston, 1999.