# Contents

**15. Scheduling in practice** **1**
*Michael L. Pinedo*
15.1. Introduction 1
15.2. Application of single machine models 2
15.3. Application of parallel machine models 4
15.4. Application of multi-operation models 6
15.5. General principles of scheduling system design 9
15.6. Schedule generation techniques in scheduling systems 9
15.7. User interfaces in scheduling systems 12
15.8. Database issues in scheduling systems 15
15.9. Scheduling systems in practice 17
15.10. Discussion 19

# 15

# Scheduling in practice

Michael L. Pinedo
*New York University*

### 15.1. Introduction

In this chapter we focus on how the results presented in the earlier parts of this book are made useful in the real world. Before we go into the applications we do have to describe a number of the differences between the deterministic models considered in this book and the scheduling problems in the real world.

In practice, there are often not just $n$ jobs. At a certain point in time there may be indeed $n$ jobs to be scheduled, but additional jobs arrive at regular or random intervals.

There may be multiple objectives, that are subject to various weights. These weights often vary from day to day, making a parametric analysis necessary.

In the real world, it is often a *rescheduling* problem that has to be solved. That is, there is already a schedule, but there is a need to reschedule, because of a random perturbation (in the form of an arrival of a rush job, or the change in the priority of a job). This is one of the reasons why, in addition to the standard objectives described in this book, one more consideration is important, namely the “robustness” of the schedule. A schedule is called robust if the schedule, when subject to a random perturbation, does not need any major changes.

In the models considered in the earlier parts of this book, it is typically assumed that the machines are available continuously. However, in real life, machines are never available at all times. There may be many reasons why a machine may become unavailable. For example, the machine may be subject to (random) breakdowns or the machine may be subject to preventive maintenance. The availability of a machine may also depend on the shift schedules of the operators.

This chapter is organized as follows. The second section describes application of single machine models and the third section considers applications of parallel machine models. The fourth section focuses on multi-operation models. The fifth section gives an overview of the procedures that have proven to be popular in the scheduling engines and schedule generators developed and under development in industry.

## 15.2. Application of single machine models

When in a given environment one machine is the bottleneck, and it is so consistently, then a decomposition procedure may be appropriate. In such a decomposition procedure there is always a module that contains a procedure for solving a single machine scheduling problem. So the scheduling of the entire environment is done by tackling the bottleneck first. Since the bottleneck determines the overall throughput, it makes sense to schedule all other machines in such a way that the life of the bottleneck is made easy.

Such a situation occurs in many industrial settings.

**Example 15.1** (A factory in the packaging industry)**.** *Consider a factory that makes carton boxes for breakfast cereals. Each order is a request for a given number of cartons (e.g., 50,000) of a certain type. There is a committed shipping date at which time the order should be delivered. The raw materials to produce the carton are available at a certain date (the raw material includes the board, the ink needed in the printing process, the dies for the cutting, etc.). This data is typically provided by a Material Requirements Planning system. In this environment different jobs often may have different priorities, implying that each job has its own weight.*

*Suppose that in the facility there is a single printing machine for printing the paper board. This printing machine feeds into various cutters, that cut the boxes from the printed sheets. After the cutting process, the material moves to the gluers that produce the final box. Suppose that this single printing machine is the bottleneck. In order to do the scheduling of the entire factory, it makes sense to schedule the printing machine first and analyze it as a single machine model. There may be sequence dependent setup times on such a printing machine.*

*Before scheduling such a bottleneck, preliminary computations have to be done that yield "local" release dates and "local" due dates for the jobs at the bottleneck. The local release dates are estimated by retrieving from a Material Requirements Planning (MRP) system the date at which the job can start its route in the factory and estimating the time it will take the job to reach the bottleneck machine (in this example, where the bottleneck is at the first stage of the process, this transit time is typically negligible). The local due dates are estimated by taking into consideration the committed shipping dates and then estimating the amount of time it takes each job to traverse the facility after it has completed its processing on the bottleneck machine.*

In the example above there may be sequence dependent setup times on the single machine that is being analyzed. These sequence dependent setup times may or may not be negligible. In the next example, however, sequence dependent setup times for sure play an important role.

**Example 15.2** (A routing problem)**.** *Consider a warehouse with a single truck that has to bring merchandise to a number of clients. The delivery of merchandise to a client represents a job and the time that the truck spends at the client's site is the processing time. There may be release dates and due dates for the processing of that job, since the client may want the goods to be delivered within a given time period. However, the time it takes the truck to travel from one client to another represent a form of sequence dependent setup time that cannot be disregarded.*

*There are a couple of possible objectives that could be minimized when scheduling (or, equivalently, routing) the truck. One is to minimize the total weighted tardiness, i.e., have the truck deliver the goods as much as possible within the intervals requested by the customers. Another objective is to minimize the total time it takes the truck to do all its deliveries. This implies that the the sum of the distances traveled (or, equivalently, the sum of the sequence dependent setups) has to be minimized. Such a problem is, of course, equivalent to the Traveling Salesman Problem.*

From these examples it appears that single-machine models of two types in particular are of interest: $1|r_j, setups|\sum w_j f_j(d_j, C_j)$ and $1|r_j, setups|\max(w_j f_j(d_j, C_j))$. There may be, in addition to a tardiness penalty, also an earliness penalty. A number of special cases of these problems have been considered in previous chapters, i.e., single machine problems with objectives $\sum w_j U_j$, $\sum w_j T_j$, and $L_{\max}$. These special cases are all unary NP-hard. The special case $\sum U_j$ is of some importance in practice as well, since one measure according to which plant managers often are judged is the percentage of on-time shipments (which is equivalent to $\sum U_j$ ).

There are a number of ways in which these problems are dealt with in practice. First, one may establish a priority rule. The priority rules of interest are, of course, more complicated than the EDD or WSPT rules described in Chapter 3 and 4. However, they typically have somewhere an EDD or WSPT flavor in them.

Second, one may attempt to apply local search procedures to these problems, i.e., either simulated annealing, tabu-search or genetic algorithms.

Third, in applied problems one may at times apply rolling horizon procedures. Since jobs come in regularly over time, it makes sense to do some form of temporal decomposition.

It is not often that one would apply branch and bound procedures to such problems in practice. The reason is the following: The inaccuracy in the data may often be in the order of 10 or 20%. It may not pay to try to do a perfect optimization and achieve a solution that is 1% better if the data are not sufficiently exact. However, if one does apply branch and bound, then many of the dominance rules that have been established in the literature for these problems are useful. For example, if $p_j < p_k$, $d_j < d_k$, and $w_j > w_k$, then it is known that, provided both jobs $j$ and $k$ are available, job $j$ must precede job $k$ (see Chapter 6).

More general single machine models have also been considered in practice. These more general models often include multiple objectives including earliness costs.

## 15.3. Application of parallel machine models

Parallel machine applications occur in practice when the bottleneck is a workcenter with a number of machines in parallel. The remaining part of the factory is again disregarded in the same way as in the previous section. Such a setting is prevalent in many industries as well.

In industrial settings these parallel machines often have different speeds or are unrelated. Unrelated machines occur in practice often: certain jobs cannot be processed on just any one of the machines in parallel but rather only on machines that belong to a specific subset (making in essence the processing times on the other machines infinity).

The local release dates of all the jobs for the parallel machine workcenter have to be estimated and so have the due dates. The due date related objectives described in the previous section are applicable here as well. This implies that $P|r_j|\sum w_j f_j(d_j,C_j)$, $Q|r_j|\sum w_j f_j(d_j,C_j)$, and $R|r_j|\sum w_j f_j(d_j,C_j)$ are problems of interest.

Consider the same factory as the one described in Example 15.1. However, suppose now that the printing stage consists of $m$ identical machines in parallel and suppose that this stage is again the bottleneck. It makes sense then to apply a decomposition procedure in which this stage is first considered as an isolated parallel machines model.

Consider the same warehouse as the one described in Example 15.2. However, suppose that instead of a single truck, there is a fleet of $m$ trucks. Again, the $n$ clients have to receive their goods, and each truck has to be assigned a number of clients. This implies that this problem is equivalent to a parallel machine scheduling problem.

The machines in a bank of parallel machines (or the trucks in a fleet of trucks) are very often not exactly identical. They may operate, for example, at different speeds. However, the processing time of job $j$ on machine $i$ may have a certain structure. For example, suppose job $j$ is associated with an order for a given quantity of items, say $q_j$. The processing time of job $j$ on machine $i$ could then be of the form

$$p_{ij} = s_{ij} + q_j/v_{ij},$$

where $s_{ij}$ represents the setup time needed at machine $i$ for job $j$ (this setup time may depend on the machine as well as on the type of job and may or may not be sequence dependent). If the setup times on machine $i$ are sequence dependent, then the setup time has to be $s_{ikj}$, i.e., if job $k$ is followed by job $j$, then machine $i$ requires an amount of time $s_{ikj}$ for setup. The $v_{ij}$ is the speed at which machine $i$ can process job $j$. The speed $v_{ij}$ may be zero; if this is the case, then machine $i$ cannot process job $j$ (possibly for technological reasons).

**Example 15.3** (An airport terminal)**.** *Consider an airport terminal with $m$ gates. The gates are not all identical. Some are so close to others that they are not capable of serving wide-body planes. Some gates are so close to the terminal that planes have to be towed in, implying an additional setup time.*

*The arrival of a plane at the terminal is equivalent to the release date of a job. The deplaning of arriving passengers, the servicing of the plane and the boarding of the departing passengers constitute the job. This job has a processing time that is subject to a certain amount of randomness. The due date is the scheduled departure time of the plane and the completion time is the actual departure time of the job.*

*Jobs have to be assigned to machines in such a way that the total weighted tardiness is minimized. The weight $w_j$ of job $j$ depends on ......*

**Example 15.4** (A hotel)**.** *Consider a hotel with $m$ rooms. There are $n$ customers arriving over a given horizon and customer $j$ has an arrival date $r_j$ and a departure date $d_j$. The sojourn time of customer $j$ is $p_j = d_j - r_j$. If the hotel decides to give the customer a room for that period, then the hotel makes a profit $w_j$. The objective of the hotel is to maximize its profit. It is clear that the scheduling problem the hotel faces is equivalent to $P|r_j|\sum w_j U_j$.*

*This particular problem, where there is no freedom in the timing of the processing (i.e., $p_j = d_j - r_j$ ), is often referred to as a fixed interval scheduling problem, or simply an interval scheduling problem.*

Even in the case of a hotel it is typical that the "machines", i.e., the hotel rooms, are not all identical. There are singles, doubles and suites (with or without a view). So any given customer cannot just be assigned to any room; he may only be assigned to a room that belongs to a given subset. However, the "processing time" of the customer, i.e., his length of stay, does not depend on the room, i.e., in this case the machines do not have different speeds.

One objective that is also important in the parallel machine setting is the $C_{\max}$ objective. Minimizing the makespan in essence balances the workload over the machines and balancing the workload enhances the capacity utilization. Workload balance is often an important objective in practice. (Of course, this objective did not play a role in the single machine models). As we have seen in Chapter 9 a very useful rule for assigning jobs to machine is based on the LPT heuristic. The next example shows an application of the LPT rule.

**Example 15.5** (A manufacturing facility for printed circuit boards)**.** *This example originated in a manufacturing facility of IBM in the U.S. The facility has a flexible flow line for the insertion of components onto printed circuit cards. A card is transported through the line on a "magazine", which contains 100 cards of the same type. From the scheduling point of view, such a magazine with a 100 cards is equivalent to one job. A job visits up to three banks of machines. The first bank consists of the* *so-called "DIP" inserters, the second bank of the so-called "SIP" inserters, and the third bank consists of a couple of robots. Each machine has a buffer that can hold one job, but additional storage is available, if necessary. Each job must be processed by at most one machine at each bank, but some jobs may skip some banks. Each job must visit the three banks in the same order. The assignment of the jobs to the specific machines at the various banks has to be done in advance, i.e., it is not possible to have the jobs arrive at a bank, join a single queue, and, when the job is at the head of the line, take the machine that becomes available first.*

*There are a number of objectives. First, one goal is to minimize the time to complete one day's production. This objective is basically equivalent to minimizing the makespan in a multi-processor flow shop. A second objective is to minimize the queueing in the buffers, since the buffers have a finite capacity.*

*The heuristic developed for this problem consists of three modules, that have to be executed one after another. Since one of the objectives is to minimize the queueing, it is advisable that at any given bank of machines the workload is balanced over the various machines. In order to balance the workload over the various machines at any given bank, the Longest Processing Time first (LPT) heuristic is used. Recall that the LPT heuristic basically establishes an assignment of jobs to machines. After a set of jobs has been assigned to a particular machine, the sequence in which the set of jobs are processed on the machine is immaterial, i.e., it does not affect the workload balance.*

The techniques applied to these parallel machine problems in practice are very similar to the techniques applied to the single machine scheduling problems. With machines in parallel there are often also multiple objectives. The multiple objectives include due date related objectives as well as workload balancing objectives.

## 15.4. Application of multi-operation models

Multi-operation machine settings are prevalent in many industries. Actually, the examples in the single machine and the parallel machine settings were already extracted from setups that resemble multi-processor flow shops. Important applications of multi-operation models occur in the micro-electronics industries. The most important objectives in these industries are due date related, i.e., objectives typically more general than $L_{\max}$, $\sum w_j T_j$ and $\sum w_j U_j$.

**Example 15.6** (Manufacturing of printed circuit boards)**.** *Flow shops are prevalent in Printed Circuit Board (PCB) Manufacturing. The PCB manufacturing process consists of a number of steps that have to be applied sequentially to a panel that is typically made of epoxy. These steps may include:*

*(i)* *drilling of holes,*
*(ii)* *metallization*
*(e.g., processing through a copper immersion bath),*
*(iii)* *panel preparation*
*(applying Dry Film Photo Resist (DFPR)),*
*(iv)* *lamination and exposition to UV light,*
*(v)* *developing to wash away the DFPR not exposed to UV light,*
*(vi)* *electroplating of copper conductors,*
*(vii)* *chemical stripping of DFPR that has been exposed to UV light,*
*(vii)* *chemical etching of the copper layer,*
*(viii)* *stripping of the tin,*
*(ix)* *soldermask silkscreening,*
*(x)* *solder dipping,*
*(xi)* *punch pressing,*
*(xii)* *inspection.*

*A job basically consists of an order for a batch of PCB's. The batch size could be anywhere from half a dozen to several thousands. The fact that job* $j$ *is basically associated with a given quantity of boards* $q_j$ *implies that the processing times of a job at the various stages are positively correlated. The processing time of job* $j$ *at machine i could be of the form*

$$p_{ij} = s_{ij} + q_j/v_i,$$

*where* $s_{ij}$ *represents the setup time needed at machine i for job* $j$ *(this setup time may depend on the machine as well as on the type of board); the* $v_i$ *is the speed of machine i and gives an indication of how fast the machine can produce a board.*

*The main objective is typically to meet as many of the committed shipping dates as possible.*

The setting in PCB manufacturing is often a flow shop. However, if at any one of the stages there are two or more machines in parallel instead of a single machine, then the setting is referred to as a multi-processor flow shop (or flexible flow shop, or hybrid flow shop).

Multi-processor flow shops are common in many other industries as well.

**Example 15.7** (A paper mill)**.** *Paper mills typically give rise to multi-processor flow shop models. A paper mill often has more than one paper machine (it may have 2, 3, or even more) which produce the rolls of papers. These rolls of papers may have to go to an adjacent converting facility that produces cut size paper (8.5* $\times$ *11). This converting facility has a number of cutters in parallel. The jobs that have to be processed have given committed shipping dates or due dates. The objective is to minimize some due date related penalty function.*

Actually, the problem in the real world is slightly more complicated because of the fact that the parallel machines at any one of the two stages are not exactly identical.

This implies that certain jobs can only be processed on a subset of the machines at anyone of the stages.

The paper mill example is a typical example of a multi-processor flow shop without recirculation. The next example is an example of a job shop with recirculation.

**Example 15.8** (Integrated circuit manufacturing)**.** *Job shops are common in integrated circuit manufacturing (wafer fabs). The fabrication process in wafer fabs consists also of a number of steps.*

*(i) cleaning*
*(ii) oxidation, deposition, and metallization*
*(iii) lithography*
*(iv) etching*
*(v) ion implementation*
*(vi) photoresist stripping*
*(vii) inspection and measurement.*

*However, since wafers often have various layers of of circuits one on top of another, a wafer may have to go through these sequence of operations a number of times. One could refer to this environment as a flow shop with recirculation, or, in effect, a job shop.*

*The main objective here often depends on the type of wafer fab. If the facility is geared towards the mass production of DRAMs, then maximizing throughput, and therefore minimizing setup times, is important. If the facility is designed to produce more specialized products, then meeting customer due dates is more important.*

The next example is another example of a job shop with recirculation.

**Example 15.9** (Manufacturing of nuclear control rods)**.** *A manufacturing facility that produces nuclear control rods produces rods of a special alloy about 1.5 inch in diameter and 20 feet long for controlling nuclear reactions. The manufacturing procedure goes roughly as follows. Each incoming rod is extruded four times. The prototype rod is drawn through dies four times to slowly reduce its diameter. After being extruded four times the rod is cut into pieces of the appropriate length. Each time the diameter is reduced the process is said to have made a pass. Such a pass consists of pilgering, cleaning, and then tempering in an annealing surface to reduce the stresses induced by extrusion. The pilgering and the cleaning in the different passes is done at different workstations. However, there is only one annealing furnace. So at the end of each one of the first three passes a rod has to go through the same annealing furnace. Because of the fact that that each job has to go three times through the same annealing furnace, this is an example of a job shop with recirculation. The main objective in this case is the minimization of the sum of the weighted tardinesses. So the problem can be modelled as* $J|r_j|\sum w_j T_j$.

*A number of different techniques have been developed for this problem, branch and bound techniques as well as shifting bottleneck techniques. When these tech-*

*niques are used it is usually assumed that the weight of job* $j$, $w_j$, *does not change while the job traverses the shop.*

*However, another scheduling technique that is often used in practice in such a setting requires the use of a dispatching or priority rule at each workstation. Such rules may be used at the different stations independently from one another. There is actually an interesting phenomenon that often takes place in practice when priority rules are used. The weight or priority of a job, on its route through the shop, goes up. This implies that the weight of a job that is waiting at the furnace on its third pass is higher then the weight of a job that is waiting at the furnace on its first pass. The rationale for this higher priority is based on the fact that at the end of the third pass more value has been added to the job.*

## 15.5. General principles of scheduling system design

Analyzing a scheduling problem and developing a procedure for dealing with the problem on a regular basis is, in the real world, only part of the story. The procedure has to be embedded in a system that enables the scheduler to actually apply it. The scheduling system has to be integrated into the information system of the organization, which can be a formidable task.

The information system within an organization or company typically is a very large Enterprise Resource Planning (ERP) system that serves as a backbone for the entire corporation. Every division of the organization has access to the system and many decision support systems are plugged into it, e.g., forecasting systems, inventory control systems, MRP systems, and planning and scheduling systems. The flow chart of such an ERP system is depicted in Figure 15...

A scheduling system consists of a number of different modules. Those of fundamental importance are:

(i) the schedule generation modules,
(ii) the user interface modules, and
(iii) the database, object base, and knowledge-base modules.

The interactions between these different modules are depicted in Figure 15...

## 15.6. Schedule generation techniques in scheduling systems

A schedule generation module contains a suitable model with objective functions, constraints and rules, as well as heuristics and algorithms.

Current schedule generation techniques are an amalgamation of several approaches that have been converging in recent years. One approach, predominantly followed by industrial engineers and operations researchers, may be referred to as the *algorithmic* approach. Another, that is often followed by computer scientists and artificial intelligence experts, may be referred to as the *knowledge-based* approach.

Recently, these two approaches have started to converge and the differences have become blurred. Some recent hybrid systems combine a knowledge base with fairly sophisticated heuristics. Certain segments of the procedure are designed according to the algorithmic approach, while other segments are designed according to the knowledge-based approach.

**Example 15.10** (Architecture of a scheduling system in a wafer fab)**.** *A hybrid scheduling system has been designed for a particular semiconductor wafer fabrication unit. The system consists of two levels. The higher level operates according to a knowledge-based approach. The lower level is based on an algorithmic approach; it consists of a library of algorithms.*

*The higher level performs the first phase of the scheduling process. At this level, the current status of the environment is analyzed. This analysis takes into consideration due date tightness, bottlenecks, and so on. The rules embedded in this higher level determine the type of algorithm to be used at the lower level in each situation.*

The algorithmic approach usually requires a mathematical formulation of the problem that includes objectives and constraints.

A number of generic algorithmic procedures have proven to be very useful in practice. In this section we give a very short overview of the generic procedures that are popular.

Many industrial scheduling systems use priority or dispatching rules. The most popular priority rules are WSPT, EDD, CP, etc. These priority rules are based on a simple sort and operate therefore in $O(n \log n)$. They are therefore very useful for providing a first crack at the problem.

Many procedures in practice are based on some form of decomposition. Instances in practice are typically very large, involving hundreds of jobs and tens of machines. There are several forms of decomposition. One form of decomposition is based on machine decomposition. The machines are scheduled one at a time. Every time a machine is scheduled, the machine is scheduled throughout. One famous form of machine decomposition is often referred to as the shifting bottleneck procedure. Another form of decomposition is based on a partitioning of the time axis. These forms of decomposition are often called rolling horizon procedures.

Other techniques that have proven to be very popular in practice are the local search techniques, e.g., simulated annealing, tabu-search, and genetic algorithms. These techniques have proven to be very versatile, in particular for nonpreemptive scheduling problems, and easy to code.

There are cases of scheduling in practice that can be formulated very precisely and are not subject to uncertainty and randomness. Such cases often can be formulated as integer or disjunctive programs. Such formulations may lead then to enumeration techniques such as branch and bound. If the solution of such a problem is not time constrained, then a complete enumeration may be possible. If there are constraints with regard to the computation time, then beam search may be more appropriate.

Algorithmic schedule generation may consist of multiple phases. In the first phase, a certain amount of *preprocessing* is done, where the problem instance is analyzed and a number of statistics are compiled, e.g., the average processing time, the maximum processing time, the due date tightness. A second phase may consist of the actual algorithms and heuristics, whose structure may depend on the statistics compiled in the first phase. A third phase may contain a *postprocessor*. The solution that comes out of the second phase is fed into a procedure, such as simulated annealing or tabu-search, to see if improvements can be obtained. This type of schedule generation is usually coded in a procedural language such as Fortran, Pascal or C.

The knowledge-based approach is different from the algorithmic approach 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, rules or objects are used. This approach is often used when it is only necessary to find a *feasible* solution given the many constraints or rules; however, as some schedules are ranked ”more preferable” than others, heuristics may be used to obtain a “more preferred” schedule. Through a so-called *inference engine,* the approach attempts to find schedules that do not violate prescribed rules and satisfy stated preferences as much as possible. The inferencing techniques are usually so-called *forward chaining* and *backward chaining* algorithms. A forward chaining algorithm is knowledge driven. It first analyzes the data and the rules and, through inferencing techniques, attempts to construct a feasible schedule. A backward chaining algorithm is result oriented. It starts out with a promising schedule and attempts to verify whether it is feasible. Whenever a satisfactory solution does not appear to exist or when the scheduler judges that it is too difficult to find, the scheduler may reformulate the problem through a relaxation of the constraints. The relaxation of constraints may either be done automatically (by the system itself) or by the user. Because of this aspect, the approach has been at times also referred to as the *reformulative* approach.

The programming style used for the development of knowledge-based systems is different from the ones used for systems based on algorithmic approaches. The programming style may depend on the form of the knowledge representation. If the knowledge is represented in the form of *IF-THEN* rules, then the system can be coded using an expert system shell such as *OPS5.* The expert system shell contains an inference engine that is capable of doing forward chaining or backward chaining of the rules in order to obtain a feasible solution. This approach may have difficulties with conflict resolution and uncertainty. If the knowledge is represented in the form of logic rules, then an ideal programming language is Prolog. If the knowledge is represented in the form of frames, then a language with object oriented extensions is required, e.g., LISP or C++. These languages emphasize user-defined objects that facilitate a modular programming style.

Algorithmic approaches as well as knowledge-based approaches have their advantages and disadvantages. An algorithmic approach has an edge if

- the problem allows for a crisp and precise mathematical formulation,
- the number of jobs involved is large,
- the amount of randomness in the environment is minimal,
- some form of optimization has to be done frequently and in real time, and
- the general rules are consistently being followed without too many exceptions.

A 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 only if feasible schedules are required. Some system developers believe that changes in the scheduling environment or rules can be more easily incorporated in a knowledge-based system than in a system 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.

A major disadvantage of the knowledge-based approach is that obtaining a reasonable schedule may take substantially more computer time than an algorithmic approach. In practice certain scheduling systems have to operate in near-real time (it is very common that schedules must be generated within several minutes).

The amount of available computer time is an important issue with the selection of a schedule generation technique. The time allowed for schedule generation varies from application to application. Many applications require real time performance, where the schedule has to be generated in seconds or minutes on the computer at hand. This may be the case if rescheduling is required throughout the day, due to substantial schedule deviations. It would also be true if the scheduler runs iteratively, requiring human interaction between iterations (perhaps for adjustments of workcenter capacities). However, some applications do allow for overnight number crunching. For example, the scheduler at the end of the afternoon executes the program and wants an answer by the time he or she arrives at work the next day. A few applications require extensive number crunching. When, in the airline industry, quarterly flight schedules have to be determined, the investments at stake are such that a week of number crunching on a mainframe is fully justified.

As said before, the two approaches have been converging and most recent scheduling systems have elements of both. One language of choice is now C++ as it is an easy language for coding algorithmic procedures and it also has object-oriented extensions.

## 15.7. User interfaces in scheduling systems

User interface modules are important, especially with regard to the implementation process. Without an excellent user interface there is a good chance that, regardless of its capabilities, the system will be too unwieldy to use.

The user interfaces are very important parts of the system. These interfaces may determine whether the system is going to be used or not. Most user interfaces, whether the system is based on a workstation or PC, make extensive use of window mechanisms. The user often wants to see several different sets of information at the same time. This is the case not only for the static data that is stored in the database, but also for the dynamic data that depend on the schedule.

Some user interfaces allow for extensive user interaction. The scheduler can change the current situation or the current information. Other user interfaces do not allow the scheduler to change anything. For example, an interface that displays the values of all the relevant performance measures would not allow the scheduler to change any of the numbers. The scheduler may be able to change the schedule in another interface that then automatically changes the values of the performance measures, but the scheduler cannot change the performance measures directly.

User interfaces for database modules often take a fairly conventional form and may be determined by the particular database package used. These interfaces must allow for user interaction, as data such as due dates often have to be changed during a scheduling session.

The schedule generation module may provide the user with a number of computational procedures and algorithms. Such a library of procedures within the schedule generation module will require its own user interface, enabling the scheduler to select the appropriate algorithm or even design an entirely new procedure.

User interfaces that display schedule information take many different forms. Interfaces for schedule *manipulation* determine the basic character of the system, as these are the ones used most extensively by the scheduler. The different forms of schedule manipulation interfaces depend on the level of detail as well as on the planning horizon being considered. Two such interfaces are described in detail, namely:

(i) the Gantt Chart interface,
(ii) the Dispatch List interface.

The first, and probably most popular, form of schedule manipulation interface is the Gantt chart. The Gantt chart is the usual horizontal bar chart, with the x-axis representing the time and the y-axis, the various machines. A color and/or pattern code may be used to indicate a characteristic or an attribute of the corresponding job. For example, jobs that are completed after their due date under the current schedule may be colored red. The Gantt chart usually has a number of scroll capabilities that allow the user to go back and forth in time or focus on particular machines, and is usually mouse driven. If the user is not entirely satisfied with the generated schedule, he may wish to perform a number of manipulations on his own. With the mouse, the user can "click and drag" a job from one position to another. Providing the interface with a click and drag capability is not a trivial task for the following reason. After changing the position of a particular operation on a machine, other operations on that machine, which belong to other jobs, may have to be pushed either forward or backward in time to maintain feasibility. The fact that other operations have to be processed at different times may also have an effect on other machines. This is often referred to as *cascading* or *propagation* effects. After the scheduler repositions an operation of a job, the system may call a reoptimization procedure embedded in the scheduling routine to control the cascading effects in a proper manner.

**Example 15.11** (Cascading effects and reoptimization)**.** *Consider a three machine flow shop with unlimited storage space between the successive machines and therefore no blocking. The objective is to minimize the total weighted tardiness. Consider a schedule with 4 jobs as depicted by the Gantt chart in Figure 15... If the user swaps jobs 2 and 3 on machine 1, while keeping the order on the two subsequent machines the same, the resulting schedule, because of cascading effects, takes the form depicted in Figure 15... If the system has reoptimization algorithms at its disposal, the user may decide to reoptimize the operations on machines 2 and 3, while keeping the sequence on machine 1 the way he constructed it. A reoptimization algorithm then may generate the schedule depicted in Figure 15... To obtain appropriate job sequences for machines 2 and 3, the reoptimization algorithm has to solve an instance of the two machine flow shop with the jobs subject to given release dates at the first machine.*

Gantt charts do have disadvantages, especially when there are many jobs and machines. It may be hard to recognize which bar or rectangle corresponds to which job. As space on the screen (or on the printout) is rather limited, it is hard to attach text to each bar. Gantt chart interfaces usually provide the capability to click the mouse on a given bar and open a window that displays detailed data regarding the corresponding job. Some Gantt charts also have a filter capability, where the user may specify the job(s) that should be exposed on the Gantt chart while disregarding all others.

The second form of user interface displaying schedule information is the *dispatch-list* interface (see Figure 15...). Schedulers often want to see a list of the jobs to be processed on each machine in the order in which they are to be processed. With this type of display schedulers also want to have editing capabilities so they can change the sequence in which jobs are processed on a machine or move a job from one machine to another. This sort of interface does not have the disadvantage of the Gantt chart, since the jobs are listed with their job numbers and the scheduler knows exactly where each job is in a sequence. If the scheduler would like more attributes (e.g., processing time, due date, completion time under the current schedule, and so on) of the jobs to be listed, then more columns can be added next to the job number column, each one with a particular attribute. The disadvantage of the dispatch-list interface is that the scheduler does not have a good view of the schedule relative to time. The user may not see immediately which jobs are going to be late, which machine is idle most of the time, etc.

Clearly, the different user interfaces for the display of schedule information have to be strongly linked with one another. When the scheduler makes changes in either the Gantt chart interface or in the dispatch-list interface, the dynamic data may change considerably due to the cascading effects or reoptimization routines. Changes made in one interface, of course, have to be shown immediately in the other interfaces as well.

User interfaces for the display of schedule information have to be linked with other interfaces also, e.g., database management interfaces and schedule generation interfaces. For example, the scheduler may modify an existing schedule in the Gantt chart interface by clicking and dragging; then he may want to freeze certain jobs in their respective positions. After doing this, he may want to reoptimize the remaining jobs, which are not frozen, using an algorithm in the schedule generation module. These algorithms are similar to the algorithms described in Part 2 for situations where machines are not available during given time periods (because of breakdowns or other reasons). The schedule manipulation interfaces have to be, therefore, strongly linked with the interfaces for algorithm selection.

Schedule manipulation interfaces may also have a separate window that displays the values of all relevant performance measures. If the user has made a change in the schedule the values before and after the change may be displayed. Typically, performance measures are displayed in plain text format. However, more sophisticated graphical displays may also be used.

Some schedule manipulation interfaces are sophisticated enough to allow the user to split a job into a number of smaller segments and schedule each of these separately. Splitting an operation is equivalent to (possibly multiple) preemptions. The more sophisticated schedule manipulation interfaces also allow different operations of the same job to overlap in time. In practice, this may occur in many settings. For example, a job may start at a downstream machine of a flow shop before it has completed its processing at an upstream machine. This occurs when a job is a large batch of identical items. Before the entire batch has been completed at an upstream machine, parts of the batch may already have been transported to the next machine and may have already started their processing there.

## 15.8. Database issues in scheduling systems

The database modules play a crucial role in the functionality of the system. Significant effort is required to make a factory's database suitable for input to the scheduling system.

The database management subsystem may be either custom-made or a commercial system. A number of the commercial database systems available on the market have proven to be useful for scheduling systems. These are usually relational databases incorporating *Stuctured Query Language (SQL)*. Examples of such database management systems are Oracle, Sybase and Ingres.

Whether a database management subsystem is custom made or commercial, it needs a number of basic functions, which include multiple editing, sorting and searching routines. Before generating a schedule, the scheduler may want to see certain segments of the order masterfile and collect some statistics with regard to the orders and the related jobs. Actually, at times, he may not want to feed all jobs into the scheduling routines, but rather a subset.

Within the database a distinction can be made between *static* and *dynamic* data. Static data are all job and machine data that do *not* depend on the schedule. These include the job data that are specified in the customer's order form, such as the ordered product quantity (which is proportional to the processing times of all the operations associated with the job), the committed shipping date (the due date), the time at which all necessary material is available (the release date) and possibly some processing (precedence) constraints. The priorities (weights) of the jobs are also static data as they are not schedule dependent. Having different weights for different jobs is usually a necessity, but determining their values is not all that easy. In practice, it is seldom necessary to have more than three priority classes; the weights are then, for example, 1, 2 and 4. The three priority classes are sometimes described as "hot", "very hot" and "hottest" dependent upon the level of manager pushing the job. These weights actually have to be entered manually by the scheduler into the information system database. To determine the priority level, the person who enters the weight may use his own judgement, or may use a formula that takes certain data from the information system into account (for instance, total annual sales to the customer or some other measure of customer criticality). The weight of a job may also change from one day to another; a job that is not urgent today, may be urgent tomorrow. The scheduler may have to go into into the file and change the weight of the job before generating a new schedule. Static machine data include machine speeds, scheduled maintenance times, and so on. There may also be static data that are both job and machine dependent, e.g., the setup time between jobs $j$ and $k$ assuming the setup takes place on machine $i$.

The dynamic data consists of all the data that are schedule dependent: the starting and completion times of the jobs, the idle times of the machines, the times that a machine is undergoing setups, the sequences in which the jobs are processed on the machines, the number of jobs that are late, the tardinesses of the late jobs, and so on.

The calendar function is often also part of the database system. It contains information with regard to factory holidays, scheduled machine maintenance, number of shifts available, and so on. Calendar data are sometimes static, e.g., fixed holidays, and sometimes dynamic, e.g., preventive maintenance shutdowns.

Some of the more modern scheduling systems may rely on an object base in addition to (or instead of) a database. One of the main functions of the object base is to store the definitions of all object types, i.e., it functions as an object library and instantiates the objects when needed. In a conventional relational database, a data type can be defined as a schema of data. For example, a data type "job" can be defined as in Figure ... and an instance can be as in Figure ... Object types and corresponding instances can be defined in the same way. For example, an object type "job" can be defined and corresponding job instances can be created. All the job instances have the same type of attributes.

There are two crucial relationships between object types, namely, the "is-a" relationship and the "has-a" relationship. An is-a relationship indicates a generalization and the two object types have similar characteristics. Often the two object types are referred to as a subtype and a supertype. For example, a "machine" object type is a special case of a "resource" object type and a "tool" object type is another special case of a resource object type. A "has-a" relationship is an aggregation relationship; one object type contains a number of other object types. A "workcenter" object is composed of several machine objects. A "plant" object comprises a number of workcenter objects. A "routing table" object consists of job objects as well as of machine objects.

Object types related by is-a or has-a relationships have similar characteristics with regard to their attributes. In other words, all the attributes of a supertype object are used by the corresponding subtypes. For example, a machine object has all the attributes of a resource object and it may also have some additional attributes. This is often referred to as inheritance. A hierarchical structure that comprises all object types can be constructed. Objects can be retrieved with commands that are similar to SQL commands in relational databases.

While virtually every scheduling system relies on a database or an object base, few scheduling systems have a module that serves specifically as a knowledge-base. However, knowledge-bases may become more important in the future.

The design of a knowledge-base, in contrast with the design of a database or object base, has a significant impact on the overall architecture of the system, in particular on the schedule generator. The most important aspect of a knowledge-base is the knowledge *representation.* One form of knowledge representation is through *rules.* There are several formats for stating rules. A common format is through an *IF-THEN* statement. That is, *IF* a given condition holds, *THEN* a specific action has to be taken.

Another format for stating rules is through *predicate logic* that is based on propositional calculus. An appropriate programming language for dealing with rules in this format is Prolog.

A second form of knowledge representation is through so-called *frames* or *schemata.* A frame or schema provides a structured representation of an object or a class of objects. A frame is a collection of slots and values. Each slot may have a value class as well as a default value. Information can be shared among multiple frames by inheritance. Frames lower in the hierarchy are applicable to more specific operations and resources than the more generic frames at higher levels in the hierarchy.

Just like activities, constraints and rules can be represented by schemata as well.

## 15.9. Scheduling systems in practice

The last three decades has 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, e.g., ILOG, I2, Manugistics.

These companies, whenever they landed a contract, had to customize their software to the specific applications. Since 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:

(i) shifting bottleneck procedures,
(ii) local search procedures,
(iii) mathematical programming procedures.

These companies, which at the outset were focussing 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 since 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 a 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 which at the outset specialized in ERP systems, e.g., 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 could 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), or 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, e.g., SAP, Baan, PeopleSoft, 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.

## 15.10. Discussion

From this chapter it should be clear that in practice it is extremely common that the procedures implemented are of a hybrid nature. Only the most application-specific system can rely on a very specific solution procedure. However, even those systems may need at times various solution techniques, because often instances of different sizes of the same problem have to be solved on different days requiring different approaches.

Some of the most popular techniques used in industrial systems are local search techniques. These techniques often have the important advantage that they are relatively easy to program and to implement. However, they have one disadvantage: usually they are not that easy to implement in settings where preemptions are allowed.

Industrial systems typically have one component that is very important in practice and that hardly plays any role in the theoretical models studied in the literature. That component is the calendar. The calendar is often a very large component in many scheduling systems. It plays a role in preventive maintenance scheduling, workforce (or shift) scheduling, and so on. The shift scheduling module may at times have to interact with the job (or machine) scheduling module.

There are dozens of software companies that have developed scheduling systems for the various industries. The costs of these systems range anywhere from $10,000 to $500,000. Often they require an amount of customization that costs even more.

One can make a distinction between generic scheduling systems and application-specific systems. The general architecture may be very different.

Lately academic researchers have been toying with the ideas of developing web enabled (Internet) scheduling systems. Designing scheduling systems have a number of important advantages. ...

### Notes

Pinedo (1995), Singer and Pinedo (1998), Pinedo and Chao (1999), Pinedo and Singer (1999), Yang, Kreipl, and Pinedo (2000).