in the field of scheduling, were omitted, this book would be fundamentally different; it certainly would be much shorter. Finally, I wish to thank Mrs. Barbara Vickery who single-handedly typed the manuscript. The keen wit and enormous patience she exhibited throughout this entire project have been exceeded only by her great skill and unfailing professionalism.

CHAPTER 1

INTRODUCTION

A suitable (and frequently employed) description of the subject of scheduling is that it is a field of study concerned with the *optimal allocation or assignment of resources, over time, to a set of tasks or activities.* In the so-called "real world," these resources (money, labor, machines, etc.) are generally restricted or scarce so this allocation inevitably gives rise to competition among tasks that are vying for their use. In turn, decisions regarding the resolution of these natural scheduling conflicts lead to purely combinatorial questions regarding how lists of tasks are to be arranged or sequenced. As the standard lead-in puts it, this suggests both good news and bad news.

The good news is that scheduling problems exist everywhere. Even a conservative reading of the description above leaves room enough to include important problems in the obvious settings of manufacturing, transportation, and logistics as well as ones not so evident in fields such as communications, media management, and sports. Moreover, many of these settings, and specifically the problems they exhibit, are meaningful in a practical sense. That is, effective scheduling solutions accordingly can produce substantial economic dividends. Now for the bad news.

Many scheduling problems, while mathematically challenging and thus of intellectual interest, are exceptionally difficult to solve. The inherent combinatorial explosiveness implied by the necessary requirement to examine the underlying sequencing problem present in many of these scheduling contexts should at least suggest that arriving at appropriate (*i.e.*, optimal or even near optimal) solutions might be a very tedious undertaking. In fact, this is more than a mere suggestion, for we now possess formal evidence that many scheduling problems are intractable in a particular, well-defined sense, and moreover, that this condition is likely to be a permanent one. Still, such a state of affairs, while depressing, does not render the corresponding problems unworthy of investigation. In addition, their importance is not diluted by simply knowing that they are hard, formality notwithstanding. It may, however, mean that we simply need to alter our approaches to such problems and perhaps be a little (or a lot) less ambitious in our expectations regarding their resolution.

In any event, our use of the good news-bad news theme helps to underscore what must be considered a major influence on the enormous growth of results in modern scheduling theory during the last thirty to forty years. For when combined with the myriad settings in which important scheduling problems arise, the realization that many possess mind-boggling complexity has nurtured an expanse of developments as rich and as varied as any within the so-called applications subject areas of operations research and the mathematical sciences. A key aim of this book is to not only reflect this proliferation, but to create an appreciation for it as well.

1.1 Some Scheduling Problems

In the following examples, we exhibit different types of scheduling problems. Note that our aim in these choices is not only to exhibit variety but also to convey a sense of the sorts of phenomena that accompany many such problems and that contribute directly to the frustration as well as the intellectual challenge in dealing with them.

1.1.1 Bicycle Assembly

This example is borrowed from Graham (1978). A bicycle manufacturer assembles finished parts for each bike by employing assembly teams, *i.e.*, each team consists of three workers who together are responsible for the assembly of a complete unit. There are several teams but the work of each has been standardized and so it is sufficient to examine only the activity at one work station.

The Industrial Engineering department has performed an operations analysis and the assembly of a bicycle has been broken down into 10 distinct operations. These include such tasks as frame preparation, front and rear wheel mounting/alignment, installation of the gear cluster, attachment of pedals, and so forth. Each of the 10 tasks requires a given duration or processing time that has been determined from standard time and motion analysis. In addition, the process of assembly is at least partially governed by certain technological constraints. For example, pedals can be placed on the bicycle crank only after the latter has been mounted on the frame. Overall, these technological *precedence constraints* can be captured by the pictorial representation in Figure 1.1. Tasks are denoted for convenience by $T_1, T_2, \ldots, T_{10}$, and the numbers by each represent the task's respective duration time in minutes.

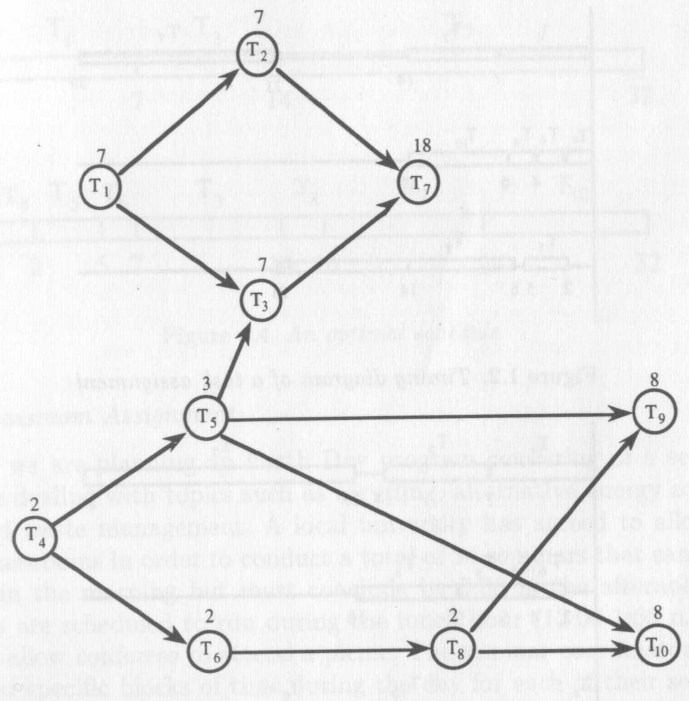

Figure 1.1. *Bicycle assembly representation*

We now come to the problem. Specifically, we seek an assignment of the tasks to the three members of the assembly team that minimizes the overall assembly time for a single unit. The team does not start on another unit until the present one is completed, and each task, once assigned, is performed only by the respective team member. In addition tasks are not preempted, *i.e.*, once started by a team member, a task is performed to completion before another one can be worked on by that member.

First, we determine if the problem possesses interest. For example, if we assign tasks T_1, T_2, and T_7 to the first worker; T_4, T_6, T_8, and T_{10} to the second; and the remainder to the third team member and if everyone performs his assigned tasks as soon as allowed by the aforementioned constraints, then the time required to complete one assembly could be as long as 39 minutes. The corresponding schedule of the tasks is displayed in a timing diagram in Figure 1.2.

On the other hand, the schedule shown in Figure 1.3 indicates that we can do better and still satisfy the precedence constraints. Indeed, the task assignment depicted has a completion time very close to that of an optimal

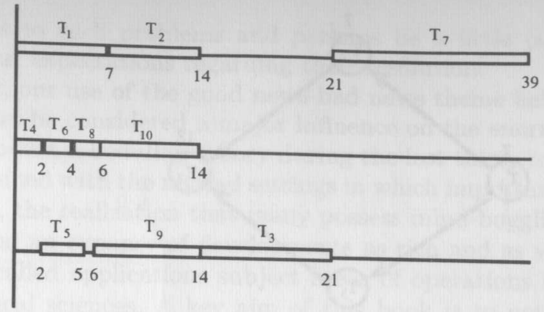

Figure 1.2. *Timing diagram of a task assignment*

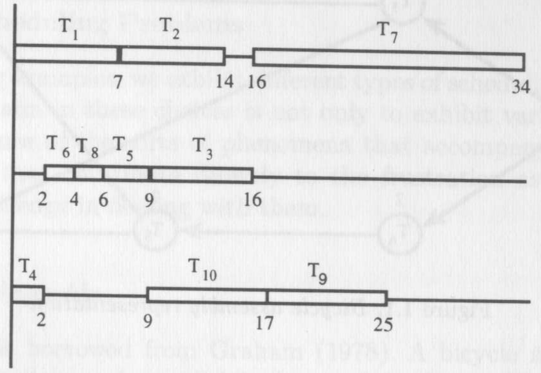

Figure 1.3. *An improved assembly schedule*

schedule. In fact, no (feasible) assignment exists that requires less than 32 minutes since the chain of tasks $T_1 - T_2 - T_7$, which cannot be overlapped, consumes 32 minutes alone.

In any event, the point to be made is that there are many task assignments possible in this case and some are certainly better than others. Moreover, the complication in finding an optimal or at least a "good" schedule for this sort of problem can only be exacerbated by increasing the number of tasks or the number of team members to which they are to be assigned.

But before leaving this illustration, we demonstrate that it is indeed possible to find a schedule of length 32 minutes and, in fact, by even employing assembly teams of reduced size! To see this, suppose we assign the 10 tasks to two team members as (T_1, T_2, T_7) and $(T_4, T_5, T_6, T_3, T_8, T_9, T_{10})$ and let us process them in the order indicated (adhering, of course, to the same technological constraints). The resultant schedule is shown in Figure 1.4.

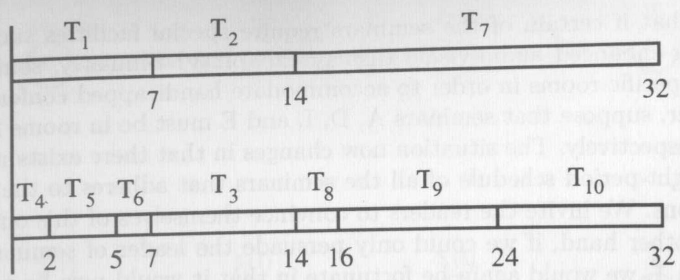

Figure 1.4. *An optimal schedule*

1.1.2 Classroom Assignment

Suppose we are planning an Earth Day program consisting of a series of seminars dealing with topics such as recycling, alternative energy sources, and solid waste management. A local university has agreed to allow the use of classrooms in order to conduct a total of 14 seminars that can begin at 8:00 in the morning but must conclude by 5:00 in the afternoon. No seminars are scheduled to run during the lunch hour (12:00-1:00 p.m.) in order to allow conferees to attend a picnic. The seminar coordinators have requested specific blocks of time during the day for each of their sessions, where the latter are denoted by the letters below. A given session must be scheduled in each of the consecutive, 1-hour periods as shown.

Seminar :	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*
Periods :	2	8	4, 5	1, 2	3, 4, 5	6	7, 8	2, 3	1, 2	5	6, 7	3, 4	8	2, 3, 4

Now, the university has offered five classrooms and would prefer that no more than this be used. The issue then is one of finding a schedule (if possible) that accommodates all of the sessions, uses no more than the stated five rooms, and that fits into the eight allowable time periods.

First, we observe that *at least* five rooms will be required since the coordinators of seminars A, D, H, I, and N requested a common period for their sessions, *i.e.*, period 2. On the other hand, this does not mean that five rooms will suffice, although in this particular instance we are fortunate, as the assignment and corresponding schedule below indicates.

Period	1	2	3	4	5	6	7	8
Room 1	*D*	*D*		*C*	*C*	*F*		*B*
Room 2	*I*	*I*	*E*	*E*	*E*		*G*	*G*
Room 3		*H*	*H*		*J*	*K*	*K*	
Room 4		*N*	*N*	*N*				*M*
Room 5		*A*	*L*	*L*				

But what if certain of the seminars require special facilities such as a computer-enhanced audio-visual display capability? Similarly, some may require specific rooms in order to accommodate handicapped conferees. In particular, suppose that seminars A, D, I, and E must be in rooms 2, 3, 1, and 4, respectively. The situation now changes in that there exists no five-room, eight-period schedule of all the seminars that adheres to the added restrictions. We invite the readers to convince themselves of this outcome. On the other hand, if we could only persuade the leader of seminar E to use room 1, we would again be fortunate in that it would now be possible to find a suitable schedule that satisfies the other requirements and that does not compel us to ask the university for another room.

While we also leave the explicit verification of the last claim above as an exercise, we do suggest that the reader be clear on the point that this sort of "classroom assignment" problem, like the bicycle assembly problem before it, possesses serious combinatorial interest. However, doubters are certainly allowed to peek at the contents of Chapter 6.

1.1.3 Scheduling Athletic Events

The final examination in a junior high school coeducational physical fitness class consists of a series of events, *e.g.*, situps, running in place, vertical jumps, etc. The boys are tested together (by event), as are the girls. The events may or may not be the same for each gender and each event is estimated to consume a fixed length of time depending upon such factors as the number of students to be tested, the type of activity required, and equipment setup needed. Only one physical education teacher is available to conduct the testing, and as a consequence only one event (for boys or girls) can be evaluated at a time. In addition to the "event duration" just specified, there is also a rest period following each event. This is a length of time the boys or girls must wait after one event is completed before the next one can begin. The length of these periods depends on the physical demands of the particular event just completed.

Suppose four events are to be evaluated for both the boys and the girls. We shall also assume the events are to proceed for both groups in a pre-specified order. Consider Figure 1.5. Here, the events are labeled relative to gender and by event number, *e.g.*, B_2 is the second event for boys while G_4 is the fourth and last event for the girls. The first number by each event specification is the event duration time and the second is the aforementioned "cooling off" time. After event four in both cases, students are free to leave and so no rest time is specified accordingly. The objective of the teacher is simple: find a schedule of the events that completes all testing as soon as possible.

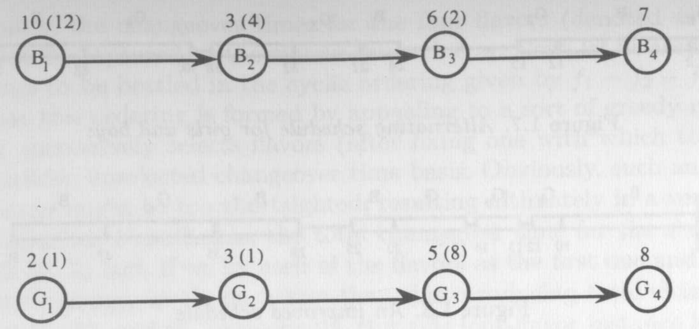

Figure 1.5. *Representation of event orders for boys and girls*

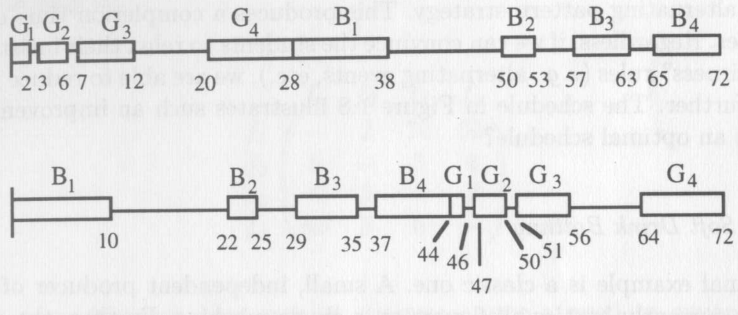

Figure 1.6. *Schedule of girls (resp. boys) first, boys (resp. girls) next*

Again, it is easy to see that the problem possesses combinatorial interest in that alternative orderings of the events give rise to alternative completion times. That is, the stated completion time objective is certainly sequence-dependent. Now, suppose the instructor decides to appeal to chivalry and schedule all the events for the girls first followed by the events for the boys. The timing diagram of the corresponding schedule is shown in Figure 1.6. Chivalry aside, it is also the case (expectedly) that scheduling the boys first followed by the events for the girls produces the same completion time (see Figure 1.6). In any event, neither alternative is particularly good, which is also expected since all rest time for the girls (resp. boys) uses up valuable testing time for the boys (resp. girls).

On the other hand, the instructor may want to invoke a more egalitarian scheduling strategy and alternate events for boys and girls. Letting girls go first and proceeding as $G_1, B_1, G_2, B_2, \ldots$, etc. yields the schedule shown in Figure 1.7, which produces an improvement.

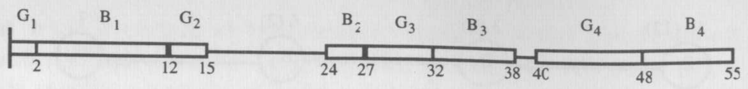

Figure 1.7. *Alternating schedule for girls and boys*

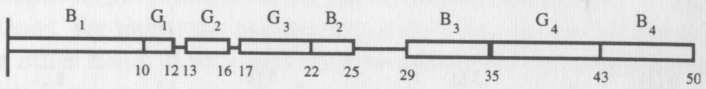

Figure 1.8. *An improved schedule*

We do slightly worse if we begin with the boys and then follow the given alternating pattern strategy. This produces a completion time of 56 minutes. Regardless, if we can convince the students to relax their insistence on "fairness" rules (*e.g.*, alternating events, etc.), we are able to reduce time even further. The schedule in Figure 1.8 illustrates such an improvement. Is this an optimal schedule?

1.1.4 Soft Drink Bottling

Our final example is a classic one. A small, independent producer of soft drinks currently bottles all flavors on a single machine. Further, the company presently serves a relatively local market by offering only four flavors. The bottling operation is repetitive in the sense that all four flavors are bottled in a sequence that repeats but the arrangement of the flavors within the cycle can be arbitrary and this leads to the problem.

While the actual bottling operation takes a fixed amount of time per flavor depending on the number of bottles to be filled of each, the machine must be cleaned after one flavor is completed and prior to the beginning of the next. This cleaning and setup time is not negligible and depends heavily on the flavor that was bottled previously. For example, particularly pungent flavors can require that the machine be cleaned and flushed with a caustic solution while substantially less preparation might be required for other flavors. Hence, two attributes of the bottling operation contribute to the overall cycle time. The first is the filling time for each flavor while the second is the cleanup or changeover time incurred between the bottling of successive flavors. The filling time is a constant, but the changeover time is a function of the sequence of flavors to be bottled. So, if the aim is to minimize cycle time, it is sufficient that we examine the equivalent problem of minimizing only the total changeover time.

Presently, the changeover times for the four flavors (denoted as f_1, f_2, f_3, f_4) can be captured by the matrix shown below. Now, let us assume the flavors are to be bottled in the cyclic ordering given by $f_1 - f_2 - f_3 - f_4$. Note that this ordering is formed by appealing to a sort of greedy mentality that successively selects flavors (after fixing one with which to start) on a smallest, unselected changeover time basis. Obviously, such an ordering strategy might be too shortsighted, resulting ultimately in a very poor overall solution. Nonetheless, the total changeover time for the stated ordering is 57. In fact, if we fix each of the flavors as the first one and repeat the given strategy, we do no better than this, producing total changeover values of 57, 83, and 66, respectively. But this four-flavor instance is small enough to examine explicitly whereupon we find that among the six possible (cyclic) orderings, $f_1 - f_2 - f_4 - f_3$ is the best, having total changeover time of only 20.

$$\begin{array}{c c} & \begin{array}{cccc} f_1 & f_2 & f_3 & f_4 \end{array} \\ \begin{array}{c} f_1 \\ f_2 \\ f_3 \\ f_4 \end{array} & \left(\begin{array}{cccc} - & 2 & 70 & 50 \\ 6 & - & 3 & 4 \\ 8 & 3 & - & 2 \\ 50 & 5 & 6 & - \end{array}\right) \end{array}$$

Suppose, however, that market changes create a need to expand the number of flavors (including a number of diet and sugar-free drinks) to 10. Certainly, the same ad hoc, "next smallest changeover" strategy described above could be used to generate a 10-flavor sequence but whatever was poor about its behavior before is not alleviated now and possibly could even be amplified. Even more obvious is that the fall-back tactic used in the four-flavor setting of listing all possible orderings and choosing the best, while certainly unsophisticated, now begins to become a serious computational question. If minimizing total changeover time is a major economic concern for the bottler, a very hard combinatorial problem now accompanies it.

1.2 Classification of Scheduling Problems

The scenarios described in the previous section provide convenient vehicles for examining the sorts of settings in which scheduling problems arise in practice. There are many from which to choose in order to provide demonstrations, but of these, many are similar, if not identical, when cast (necessarily) in an abstract context; the relevant settings are just disguises. Indeed, these abstract models give rise to a host of structures resulting in something of a "language" of scheduling which, in turn, is so rich that its own codification has been created in order to provide delineation. In this section, we give enough detail in order to employ this language effectively throughout the book.

The bicycle assembly problem is typical of a scheduling environment where a set of jobs, possibly restricted by technological precedence constraints, are to be completed by multiple processors, each of the same capability and to which any of the jobs could be assigned. That is, these processors exist in parallel and may represent machines, or in our case, people, etc. In the refined world of scheduling *theory*, we would refer to our problem as a *precedence-constrained, parallel- (sometimes, multi-) processor problem.* We would most likely need to distinguish further as to whether or not the processors are identical, nonidentical, or related; the number of processors; the value of job duration times; and even the nature of the precedence structure. As we shall observe later, variations in all of these attributes can make an enormous difference in the complexity of the basic problem.

Alternately, the soft drink bottling illustration captures the important aspect in some scheduling problems where changeover or setup times are critical, varying as functions of sequence. Rather than bottling soft drink flavors, this same phenomenon arises in settings such as paint manufacture, rolling mill operations, and scheduling changeovers for general-purpose machine tools. All of these are referred to as so-called single-processor scheduling problems subject to sequenc-dependent changeover times. In the context of our analysis, interest in whether or not soft drinks are being bottled or paint colors are being produced is secondary to the combinatorial problem of finding a correct sequence, *i.e.*, one minimizing total changeover time. Indeed, this problem is easily modeled as the well-known and much studied *traveling salesman problem.* The latter is a celebrated combinatorial optimization problem which has generated a voluminous literature of its own, including a book of the same title (see Lawler *et al.*, 1985).

Throughout this text, we will adhere to what has become the adopted convention for problem classification in the scheduling literature. Following Lawler *et al.* (1989), we employ a three-field classification $\alpha|\beta|\gamma$ where the first field corresponds to the machine or processor setting, the second to job characteristics, and the third to the optimality criterion of interest. We also will employ the following notation relating to various, integer-valued attributes of jobs (when the context is clear, we may substitute terms such as "task" or "activity" in place of "job" as the component of work). First, if a job j possesses more than one *operation*, this number is specified by $m(j)$. The duration or *processing time* of the job is τ_j when $m(j) = 1$ and τ_{jk} otherwise, where the subscript k distinguishes between operations

(performed by different machines or possibly different visits to the same machine) in the multiple-operation case. If a job possesses a *due-date* it is denoted by d_j and if the job has an assigned weight we shall use the notation w_j accordingly. In some models, jobs are not assumed available at the same (arbitrary) time, say zero, but rather each is assigned a *release time.* We denote these by r_j. Other job data that might need to be specified will be introduced as the appropriate coverage warrants.

1.2.1 Machine Environment

As indicated, the field α is for specifying the number of machines and, more crucially, for describing the configuration of the processing environment relative to the machines. If a set of single-operation jobs is to be processed on a set of *parallel identical machines*, we denote this by P. The bicycle assembly illustration falls into this category. If the machines are nonidentical but related in some way by, say, uniform machine speeds, a *uniform parallel machine* setting exists and is denoted by Q and if the processors are not related, we say that the setting is one of *unrelated parallel machines.* Here, we denote the setting by the symbol R.

When jobs possess multiple operations and if an ordering is imposed on these operations, two cases arise. If the ordering is the same for each job, a *flow shop* problem results denoted by F and if the ordering is allowed to be different among jobs, we have a *job shop*, specified by J. If there are multiple operations but no ordering on the machines is imposed, an *open shop* results which we denote by O.

When none of these configurations is relevant, we have a *single-machine* problem that is recognized by fixing the parameter for the number of machines at 1. In the cases of multiple machines but when the number is fixed, such specifications are denoted accordingly (*e.g.*, $P2, F3$, etc.) and when no fixed machine number is given, the interpretation is that this attribute is taken to be a variable and is considered as part of the problem *instance*, *i.e.*, it does not play a role in specifying a particular problem *type*.

1.2.2 Job Characteristics

The second field is reserved for attributes of jobs that when explicitly specified also figure prominently in categorizing what can result as varied and numerous outcomes in a problem's solvability status. Generally, four such attributes are recognized: whether or not a job can be interrupted or preempted during processing, denoted by *pmtn*; whether or not a precedence ordering is imposed on the jobs, denoted by *prec* (if this precedence structure is not arbitrary but restricted to take on a particular form, we will define and specify it accordingly); whether or not job-dependent release times are given; and finally, specifications regarding job duration times, *e.g.*, all jobs possess *unit duration times* ($\tau_j = 1$), $\tau_j \in \{1, 2\}$, etc. As before, an unreferenced attribute is intended to signify that the relevant condition is not imposed. For example, no reference to duration times implies that such values are arbitrary; no specification for *prec* is taken to mean that jobs are not constrained by any precedence structure.

In addition, we will generally assume that a job (resp. operation, task, etc.) starts as soon as possible subject only to natural delay caused by precedence or by processing capacity. With regard to the latter, we will assume a capacity that limits processing to at most one job at a time. Then as soon as a processor becomes available and if a job's technological predecessors (if any) have been completed, the job starts; no superfluous idle time on a processor is created. Schedules formed in this way are often referred to as *semi-active*. Important to note (and easy to see) is that when operating in this context, the distinction between "sequences" and "schedules" is eliminated.

1.2.3 Optimality Criteria

The third and final field is used for specifying measures of performance and, as a consequence, is generally self-evident. We need only be clear on notation. The *completion time* of a job is denoted by c_j and so if the measure of performance is to minimize a schedule's total completion time the entry in the third field would be Σc_j. On the other hand if the aim is to minimize the completion time of all jobs, we are really seeking to minimize the maximum completion time (sometimes called *schedule length* or *makespan*). This is denoted as $C_{\max}$.

The *lateness* of a job is given by L_j and is measured as $c_j - d_j$. When only nonnegative lateness is important, our measurement pertains to job *tardiness* denoted by T_j and is given as $\max(0, L_j)$. Total lateness or tardiness as well as maximum values for each are defined in the obvious manner and are denoted as $\Sigma L_j, \Sigma T_j, L_{\max}$, and $T_{\max}$ respectively. Other useful measures of schedule performance exist as well and will be defined when the need arises.

Relative to schedule performance generally (and where most relevant, *i.e.*, **Chapters 3, 4,** and **5**), our interest will remain confined to *regular measures*. Formally, these are measures expressible as nondecreasing functions f of job completion times such that

$$f(c_1, c_2, \ldots, c_n) < f(c_1', c_2', \ldots, c_n') \Rightarrow c_j < c_j'$$

for at least one j. The aim, naturally enough, is to produce schedules that exhibit minimum or near-minimum values of a given regular measure. Happily, the class of regular measures of schedule performance is quite rich including many that square with legitimate aims often arising in practical contexts. It is easy to see that the measures suggested above are all regular.

To illustrate overall, the problem denoted by $P3|prec|C_{\max}$ is a three-machine, parallel-machine problem with all machines identical and where the objective is to produce a minimum length schedule subject to precedence constraints on the order of job processing. This is exactly the bicycle assembly problem described earlier. On the other hand, a specification of the form $1||T_{\max}$ refers to a single-processor problem without precedence restrictions where the maximum job tardiness is to be minimized. The classic, minimum makespan, two-machine flowshop problem would appear as $F2||C_{\max}$.

1.3 Outline of the Book

In **Chapter 2** we provide background material the intent of which is to facilitate accessibility regarding the scheduling results that follow. The assumption underlying this aim is that a so-called "average user" of the book will find the coverage helpful (to be sure, some users may find it necessary while others, of course, can skip it altogether). While the heaviest treatment in the chapter pertains to the subject of *computational complexity*, we also provide enough coverage of standard notions in *graph theory* in order to more easily present and deal with existing results. We conclude the chapter with a modest discussion of *branch-and-bound*.

Our coverage of scheduling results commences in **Chapter 3** and proceeds in rather standard fashion beginning with single-machine problems. In **Chapter 4**, we consider parallel-processor models while in **Chapter 5**, flow shop, job shop, and more general models are examined. Scheduling problems that in some sense tend to resist our convenient classification format are addressed in **Chapter 6**, which includes models of the classroom scheduling problem described earlier, staffing or workforce scheduling problems, and last, timetabling problems. **Chapter 7** takes up issues in the area of *project scheduling*. Topics included are ones related to issues in activity network construction, basic scheduling calculations, and problems of time-cost optimization. We conclude with **Chapter 8** dealing with traversals. Considered are Eulerian and Hamiltonian traversals and their practical manifestations in the form of the Chinese postman and traveling salesman problems.