Chapter 1

Introduction to Flexibility and Robustness in Scheduling

1.1. Scheduling problems

A large variety of scheduling problems are to be found in many domains. Almost every sector is concerned by scheduling problems in the broad sense:

– Industrial production systems: problems may need to be solved simultaneously in machine scheduling and vehicle dispatching (automated guided systems, robotic cells, hoist scheduling problems), in workshop layout problems or supply chain management problems.

– Computer systems: for example, to make full use of the processing power provided by parallel machines or when scheduling tasks with resource constraints in real-time environments.

– Administrative systems: appointment scheduling in health care sector, general resource assignment, timetabling, etc.

– Transportation systems: vehicle routing problems, traveling salesman problems, etc.

In all cases, for a realization being described as a series of interdependent tasks, it is necessary to coordinate the implementation of these tasks, i.e. to allocate resources

Chapter written by Jean-Charles BILLAUT, Aziz MOUKRIM and Eric SANLAVILLE.

to tasks and set their execution dates. Sometimes a schedule simply consists of a sequence of tasks by machine, coupled with a simple rule for calculating task start times (for example earliest schedules). However, in the more general case it is necessary to allocate a start time to each task in the schedule definition.

The basic data of a scheduling problem (see for instance [BRU 07]) are: the tasks to schedule with their precedence constraints, their duration, resources that are necessary for their execution and a function to optimize.

Methods for solving scheduling problems draw from all the techniques of combinatorial optimization, whether approximate methods (greedy algorithms, local search, genetic algorithms, etc.) or exact methods (mathematical programming, branch-and-bound methods, dynamic programming, decomposition methods, constraint programming, etc.). Solving a particular problem may require the use of modeling tools for complex systems (simulation, Petri nets, etc.), thus leading to the definition of matchings between these methods.

The scheduling problems addressed in this book are described according to the classification schemes proposed in [GRA 79]. The scheduling problems are specified using a classification in terms of three fields, $\alpha|\beta|\gamma$ where α specifies the machine environment, β the operation characteristics, and γ the criterion to optimize.

1.1.1. *Machine environments*

The majority of scheduling problems correspond to a number of fundamental theoretical models. We have to schedule a set of n tasks or n jobs. The machine environments are specified in the field α separated into two subfields $\alpha_1\alpha_2$. Depending on the values of α_1, we may distinguish the following models:

– Single machine problems. Each task T_j of duration p_j runs on a dedicated machine that cannot handle more than one task at a time. In that case the field α_1 is absent and $\alpha_2 = 1$.

– Parallel machine problems. The tasks are to be executed on machines in parallel, and p_{ij} denotes the execution time of T_j on machine M_i:

- If $\alpha_1 = P$, the machines are identical: $p_{ij} = p_j$ for any machine M_i.
- If $\alpha_1 = Q$, the machines are uniform: $p_{ij} = p_j/s_i$ where s_i is the processing speed of machine M_i.
- If $\alpha_1 = R$, machines are unrelated: $p_{ij} = p_j/s_{ij}$ where s_{ij} is the processing speed of task T_j on machine M_i.

– Shop problems. In this model, a shop consists of m different machines. We consider a set of jobs that need to be performed. Each job J_j is described by n_j tasks (which are called *operations*). Operation J_j running on machine M_i is denoted O_{ij}, and its duration is p_{ij}. Operations belonging to the same job cannot be carried out simultaneously. There are three main types of shop:

- Flow-shop. Each job consists of m operations and the order of execution on different machines is the same for each job. In this case $\alpha_1 = F$.
- Job-shop. The number of operations is not necessarily the same for each job, and every job has its own order of execution on the machines. In this case $\alpha_1 = J$.
- Open shop. This is the least constrained shop scheduling problem. The number of operations is not necessarily the same for each job, and the order of execution on the machines is completely free. In this case $\alpha_1 = O$.

– Project scheduling under resource constraints. In this model, known as the "*resource constrained project scheduling problem*" (RCPSP), we consider a set of tasks or activities. The execution of each task T_j requires the use of a fixed amount R_{ij} of resource i. The maximum capacity of each resource i is available. The field α_1 takes the value PS ("*Project Scheduling*"). Note that the case where the capacity is unlimited corresponds to the central problem in the well-known PERT scheduling model.

If α_2 is a positive integer, the number of machines or resources is assumed to be constant. If the field α_2 is absent then this number is assumed to be arbitrary.

1.1.2. *Characteristics of tasks*

The field $\beta = \beta_1\beta_2\beta_3\beta_4$ describes the task characteristics.

Preemption means that the execution of an operation or a task can be interrupted and completed later, either on the same machine or on another machine. An operation or task can be interrupted several times. If preemption is allowed then $\beta_1 = pmtn$, otherwise field β_1 is missing.

Precedence constraints are represented by a directed graph $G = (X, \prec)$, where X denotes the complete set of tasks. $T_j \prec T_k$ means that the task T_j must be fully completed before the task T_k begins. Whether the graph G is arbitrary, a union of paths, an out-tree or an in-tree, β_2 takes the value $prec$, $chain$, out-$tree$, or in-$tree$, respectively. When there are no precedence constraints this field is missing. In the context of parallel computing (message passing) or shop management (part transfer), a quantity c_{jk} may be associated with any precedence $T_j \prec T_k$. If T_j and T_k are performed on two different processors (or machines), c_{jk} corresponds to the shortest delay between the end of T_j and the beginning of T_k, otherwise this delay is zero.

The release dates of tasks (earliest start times) are not necessarily identical. In this case, $\beta_3 = r_j$. If all tasks are assumed to be available at time 0, the field β_3 is missing.

If $\beta_4 = d_j$, we hope that the completion time C_j for each task T_j will be less than or equal to d_j, called the *due date* of T_j. If C_j exceeds d_j, the task is considered late.

1.1.3. *Optimality criteria*

When a schedule is fixed, the following variables can be computed for each task T_j or every job J_j:

– end date of the task T_j or job J_j, or completion time, noted C_j;

– lateness $L_j = C_j - d_j$ or tardiness $T_j = \max\{0, C_j - d_j\}$;

– unit penalty $U_j = 0$ if $C_j \leq d_j$, otherwise $U_j = 1$;

– flow time $F_j = C_j - r_j$.

Optimality criteria are functions to minimize. Usually, they integrate the above variables in the form of a maximum function or a sum function, possibly weighted. For example:

– the duration of the schedule or *makespan* is the function $C_{\max} = \max_{1 \leq j \leq n} C_j$;

– the weighted number of late tasks is the function $\sum_{j=1}^{n} w_j U_j$.

Optimizing a single criterion is sometimes not sufficient, and in order to solve the problem, several conflicting criteria must be taken into account. For example, a company might want to minimize delivery delays and also to minimize its storage costs. These two criteria are clearly antagonistic, and multicriteria optimization methods are required to develop a procedure which will provide the best compromise solution [T'K 06].

We conclude this section by defining three scheduling classes:

– a schedule is said to be *semi-active* if no task can be performed earlier without changing the order of execution or violating the constraints;

– a schedule is said to be *active* if no task can be performed earlier without violating the constraints;

– a schedule is said to be *without delay* if at any time t resources are not present in sufficient quantity to start an available job processed later in the schedule.

Figure 1.1 shows a single machine scheduling problem involving two tasks T_1 and T_2 with $p_1 = 2$, $p_2 = 1$, $r_1 = 1$ and $r_2 = 0$. The schedule shown in Figure 1.1a is semi-active but is neither active nor without delay, whereas the schedule shown in Figure 1.1b is semi-active, active and without delay.

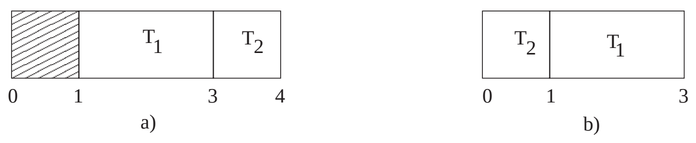

Figure 1.1. *Examples of semi-active and delay-free schedules*

Figure 1.2 shows a parallel machine scheduling ($m = 2$) of three tasks T_1, T_2 and T_3 with $p_1 = 2$, $p_2 = 2$, $p_3 = 4$, $r_1 = 1$ and $r_2 = r_3 = 0$. The schedule shown in Figure 1.2 is both active and semi-active but not without delay.

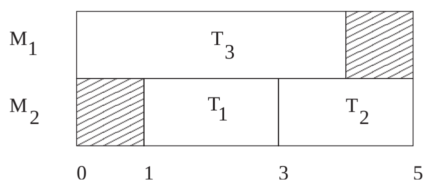

Figure 1.2. *An example of an active schedule that is not delay-free*

1.2. Background to the study

The subject under consideration is scheduling and the problem addressed in this book is the integration of flexibility and robustness in scheduling problems.

Scheduling problems are widely discussed in the literature, in a large variety of contexts (see section 1.1). We distinguish here two major classes of approach:

– Classical deterministic methods, which consider that the data are deterministic and that the machine environment is relatively simple (disjunctive resources, possibly in multiple copies: see section 1.1.1). Some traditional constraints are taken into account (precedence constraints, release dates, due dates, preemption, etc.). The criterion to optimize is often standard (makespan). Problems have been investigated and classified according to their computational complexity. A number of methods have been proposed (exact methods, greedy algorithms, approximate methods, etc.), depending on the difficulty of a particular problem. These kinds of studies are the most common in the literature devoted to scheduling problems, and there are many books dealing with the most classic problems (see for example [BLA 01, BRU 07, PIN 01]).

– On-line methods. When the algorithm does not have access to all the data from the outset, we say that the data become available step by step, or "on-line". Different models may be considered here. In some studies, the tasks that we have to schedule are listed, and appear one by one. The aim is to assign them to a resource and to specify a start time for them. In other studies, the duration of the tasks is not known in advance. These problems have given rise to many theoretical studies (e.g. [SGA 98, FIA 98]).

Flexibility occurs at the boundary between these two approaches: some information is available concerning the nature of the problem to be solved and concerning the data. Although this information is imperfect and not wholly reliable, it cannot be totally ignored. We also know that there will be discrepancies, for a number of reasons, between the initial plan and what is actually realized. Given that disruptions will occur and unforeseen circumstances arise, the aim is to propose one or more solutions that adapt well to disruptions, and then produce reactive decisions in order to ensure a smooth implementation. Another parameter here is the freedom left to the scheduler about the set of solutions it might be possible to propose: this flexibility is *internal* to the individual problem. Hence there are two kinds of flexibility, this internal flexibility, and the *chosen* flexibility, that the method really use when proposing a set of solutions (see section 1.4).

Robustness refers to the performance of an algorithm in the presence of uncertainties. Measures of robustness are required, which we will show later. Robustness can be defined at several levels: we can speak of the robustness of a solution of course, but also of the robustness of a procedure or of a conclusion [ROY 02]. Robustness is a qualifier which generally refers to a capacity to tolerate approximations (on the assumptions, model or data [ROY 02]). It is also a measure of the result after the application of a procedure in the presence of uncertainties, or after the appearance of uncertainty, for example relative to the operation duration the transport time, the availability of the most qualified personnel, etc. It is the performance characterization of an algorithm (or a complete process of schedule construction) in the presence of uncertainties (see section 1.5).

1.3. Uncertainty management

This section summarizes the sources of uncertainty for scheduling problems and shows that all data may be concerned. The different models that take into account these uncertainties are presented and the different approaches proposed in the literature are then reviewed. Recent literature is too rich to make a complete state-of-the-art survey possible. We restrict ourselves to some basic works, leaving the more specialized studies in the bibliographies of the different chapters.

The book by Kouvelis and Yu [KOU 97] presents the sources of uncertainty in operations research, particularly in scheduling, and provides a discussion on models (see also the article by Daniels and Kouvelis [DAN 95]). The article by Davenport and Beck [DAV 00] is a very detailed review with a classification of possible approaches, while Herroelen and Leus [HER 05] focus on project scheduling and describe a wide range of methods.

1.3.1. *Sources of uncertainty*

The data associated with a scheduling problem are the processing times, occurrence dates of some events, some structural features, and the costs. None of this data is free from factors of uncertainty.

The duration of tasks depends on the conditions of their execution, in particular on the necessary human and material resources. They are thus inherently uncertain, regardless of contingent factors that may impair their execution. At any time, communications between two tasks depend on the state of the communication network, the level of contention, the availability of links, and so on. Similarly, transportation times for components between separate operations in a production process will depend on the characteristics of the transportation resources available. Finally, in a production context, some resources such as versatile machines require a reconfiguration time between operations. This time depends on the type of tools needed and the location of these tools in the shop, not to mention the operator carrying out the reconfiguration.

The start times for some events within a schedule can be part of the initial data. This is the case for the arrival of a task (release date), which often depends on events outside the studied system, such as events in the supply chain or a customer order. The same is true of the due date of a task. The periods of availability of human or machine resources is also difficult to predict precisely, due to maintenance, delays or unforeseen absences of an operator or a raw material.

More radically, some events can be totally unforeseen and change the structure of the problem and consequently the ongoing schedule. A task can be added or removed without warning. The characteristics of a task can be changed, like its way of execution (regarding, for example, the range of products or the enforcement of a particular operator) or its relationships with other tasks, such as precedences or disjunctions. A machine may fail or suddenly become useless for unforeseen reasons.

Finally, if a cost is associated with a task, it can be changed without notice, especially when the considered system is part of a larger hierarchical system: the priorities are set at a higher level.

Thus, no data can be regarded as immutable, although the possibility of a change depends on the context. We can consider two cases for each piece of data (duration, date or cost): either its value is uncertain, that is to say it may take any value inside some fixed set; or its value may be subject to a disturbance, meaning that it is set in order to ensure normal functioning of the system, but can be changed by some unexpected event. This is of course always the case for structural data, which are modified according to contingent events.

1.3.2. *Uncertainty of models*

It follows from the above discussion that non-deterministic models are essential for solving concrete problems in scheduling, because of the inherent uncertainty in the data. Let us first consider the hypothesis of randomness which has given rise to a longstanding branch of research: stochastic scheduling. Here, all data (durations and also the dates of events, including possible disruptions) are modeled using random variables, and possibly constants. The probability of events is assumed to be known. From this stochastic model, it is theoretically possible to compute *a priori* the best schedules, or rather (in the case of possible disruptions) policies, i.e. the most successful decision sets (see the chapter by Weis [WEI 95] in [CHR 95] for a presentation of stochastic scheduling, as well as the book by Pinedo [PIN 01]).

This assumption of randomness is not always made, for at least three reasons. First, *a priori* knowledge about the data is not always sufficient to deduce the laws of probability associated with it, especially if the problem is addressed for the first time. Secondly, assumptions regarding independence are rarely justified: a major source of disruptions may often result in a number of uncertainties concerning various data. Finally, even if a stochastic model can be envisaged, it is often too complex to be usable.

Data values are therefore often regarded as “simply” uncertain. However, it is usually possible to maintain values within some limits, in almost all cases within a set which is discrete or continuous (interval). In the case of discrete sets, we obtain a finite but potentially large number of scenarios (a value is assigned to all data). It may be possible to allocate a probability to each scenario, even if it is not exactly computable, thus indirectly achieving a stochastic model. Even in the continuous case, it is possible to proceed using these intervals. The theory of fuzzy sets is applicable here, the application of which to scheduling problems has seen some recent developments. This is not addressed in this book, but has been the subject of a book published by Hapke and Slowinski [SLO 00]; see also Dubois *et al.* [DUB 03].

It may happen that the data are outside the considered sets. One simple solution to this is not to propose a set at all. Nevertheless, a commonly-used technique is to assign to each piece of data a central value, its estimate, and all these estimated values may then be used while anticipating the possible differences at the execution step.

To sum up, the data can be represented as either random variables (stochastic model), real intervals (interval model) or discrete sets (scenario model). They may or may not be associated with an initial estimate. It is of course possible to combine different modes of representation!

1.3.3. *Possible methods for problem solving*

We now look at different methods for solving a scheduling problem with uncertainties. The choice depends of course on the chosen model. Let us first list the steps needed to solve such a problem.

1.3.3.1. *Full solution process of a scheduling problem with uncertainties*

Obtaining a complete solution to the problem requires the following steps:

– Step 0: defining a static problem. The definition includes, in addition to the classical specifications in deterministic scheduling, the specifications of uncertainties and their modeling. The concept of schedule quality must also be specified at this stage.

– Step 1: computing a set of solutions, i.e. a family of feasible schedules achievable by a static algorithm α (static phase). A set of solutions can be obtained from a single solution, for example when the start times of some tasks may vary within a known interval.

– Step 2: during execution, a unique solution is calculated, that is to say the schedule actually carried out, which is the outcome of applying a dynamic algorithm δ (dynamic phase) to the set of possible solutions.

The solution methods differ depending on the choices made in steps 1 and 2, and therefore on the static and dynamic algorithms chosen. These choices depend, of course, on the models in step 0, which were discussed above, apart from the notion of quality, which we examine in section 1.5.

1.3.3.2. *Proactive approach*

In this approach, the focus is on step 1: knowledge of the uncertainties is used by the static algorithm to build one baseline schedule or a family of schedules. This family can be described either explicitly or implicitly. In the literature we also encounter the term *predictive approach*, the difference being that the schedule constructed in the latter case, using a static algorithm, does not take uncertainty into account. The starting point of this book is that it *must* be taken into account, and therefore we shall only look at proactive approaches.

In both approaches, predictive and proactive, step 2, during the execution, does not require any calculations: according to the real value of data, the baseline schedule is used or it is adjusted to remain feasible. These choices or adjustments are made using simple rules, such as waiting for a task to complete if it is overdue, or taking a particular action when a particular event occurs.

A stochastic model can give rise to a proactive approach, the uncertainty being taken into account in the computation of a baseline schedule.

1.3.3.3. *Proactive/reactive approach*

It is natural to couple a proactive approach, when it proposes a family of schedules, with a more elaborate step 2: as knowledge of the actual data values is acquired, and possibly after a disruption, a non-trivial dynamic algorithm is used to choose among the schedules selected in the previous (static) step those that prove to be the most efficient. This approach, responding to actual conditions while using the results of step 1, is called proactive/reactive.

In addition, let us note that it is often impossible to take all uncertainties into account, in particular disruptions, during the static phase. The example of machine failure is the most obvious but not the only one. The dynamic algorithm is then a necessity.

There are two extreme types of dynamic algorithms. The first type attempts a repair, trying to recover as fully as possible the baseline schedule or one of the selected schedules. In contrast, the second type carries out a re-optimization or post-optimization, i.e. it calculates, on the basis of actual conditions, a new schedule without further reference to the results of the static phase. The re-optimization is needed especially in the case of contingencies which significantly change the data of the initial problem.

1.3.3.4. *Reactive approach*

In the purely reactive approach, the choices specifying a schedule are made during the dynamic phase. We must keep in mind that depending on the context, the reaction time required may vary from several days for some projects to less than a second for computer applications or embedded systems. If we have relatively accurate information regarding the value of data in the ongoing schedule (see step 0), the ideal scenario is to compute the optimal decision at every point where a choice can be made, which corresponds to post-optimization, here used in an iterative manner. However, simple decision rules are more often applied, such as giving priority to tasks with smaller margins. These rules rarely build optimal schedules. In the particular case of stochastic models, it is sometimes possible to show that one set of rules (here called policy) is the best, for example according to the criterion *expectation* (see section 1.5).

Finally, when very few assumptions are made about the data (no estimates), decisions to be made at each moment are very difficult to evaluate *a priori*, which leads us into the area of *on-line scheduling* mentioned earlier. A typical case is when the characteristics of a task (duration or mode of execution) are unknown until the job is ready to be executed. The on-line scheduling reviewed by Sgall [SGA 98] is beyond the scope of this book.

1.4. Flexibility

The introduction of flexibility into a scheduling problem reflects the degree of freedom during the implementation phase of the scheduling. This flexibility can take several forms:

– Time, or *temporal* flexibility, i.e. regarding the starting times of operations. This flexibility can be seen as implicit in scheduling, since it allows some operations to drift over time, if conditions dictate. This is the first level of flexibility in scheduling.

– Flexibility regarding order of execution, or *sequential* flexibility. This means being able to change the order in which the operations should run on the machines, and implicitly presupposes temporal flexibility. It can be proposed during the execution of the sequence, allowing some operations to overtake others, if the conditions require it.

– Flexibility in assignments. In cases when there are multiple copies of resources, this allows a task to be executed using a resource other than that which was initially planned. This flexibility is a great help, for example when a machine becomes unavailable. It implicitly presupposes sequential flexibility and temporal flexibility.

– Flexibility in the execution mode. The execution mode encompasses the possibility of preemption, overlap, changes in product range, whether set-up time is taken into account, changes in the number of resources required to perform an operation, and so on. This flexibility can be proposed depending on the context to overcome a difficult situation.

Flexibility, which is a degree of freedom available during the operational phase, can be harnessed in step 1, during the static phase. Indeed, some methods, in order to give more flexibility regarding start times, will allocate the available margins to operations in proportion to their length, for example, or will allocate margins to the operations considered as the most critical, as is the case with the concept of buffer in the critical chain [GOL 97, HER 01]. In order to give more sequential flexibility, the concepts of groups of swappable tasks ([ART 99, ART 05]) and of partial order between tasks [ALO 02, WU 99, MOU 99] have been proposed. These methods were designed to build robust schedules.

The challenge when introducing flexibility is finding a way of measuring the level of flexibility obtained. Some approaches rely on measuring *a posteriori* the utility of the flexibility proposed by comparing the quality of a flexible solution to that of a non-flexible solution in the presence of disturbances. It is consequently the robustness measure that should indicate whether or not a particular flexible solution is better than a non-flexible solution.

1.5. Robustness

It is really difficult to give a unique definition for robustness, as this concept is differently defined in several domains. Furthermore, often in the literature, the definition often remains implicit in the literature or is determined by the specific target application. Finally, most authors prefer to use the concept of robust solution (and here, of robust schedule).

Let us first propose some consensus definition: a schedule is *robust* if its performance is rather insensitive to the data uncertainties. Performance must be understood here in the broad sense of solution quality for the person in charge; this naturally encompasses this solution value relatively to a given criterion, but also the structure itself of the proposed solution. The *robustness* of a schedule is a way to characterize its performance.

Anyway, analysis cannot restrict itself to one solution. We are mainly interested in the performance of the process previously detailed (see section 1.3.3.1) to build these solutions according to the real problem data. Throughout this section, the term *method* will be used to designate the whole building process of the final schedule. Thus, we follow Bernard Roy [ROY 02] who states that the person in charge is not interested in a specific solution, but in the set of solutions a method can build according to the real data, and by their variability. The questions raised in this work, in the larger framework of decision aid are fundamental also for scheduling under uncertainties. When the whole method is not explicit, but one stage alone (static or dynamic) is under study, it is then legitimate to use the terms of robust algorithm, or even of robust schedule.

The curious reader might also find it very useful to look for works about robust optimization at large, reflecting the diversity of the approaches; see Kouvelis and Yu [KOU 97], Ben-tal and Nemirovski [BEN 99], Aïssi *et al.* [AÏS 07], among many others.

In order to characterize robustness, different tools that might be used are presented. This in fact implies that several types of robustness exist. The following notations shall be used:

– $\mathcal{P}$: a static problem, together with uncertainties description. Hence $\mathcal{P}$ is a set, possibly infinite, of instances of the deterministic problem.

– $\mathcal{I}$: an effective instance of $\mathcal{P}$ (describing the effective conditions met during execution) also called the scenario.

– S: an effective schedule, obtained by the studied building process. S varies according to the considered scenario and in cares of possible ambiguity it is noted $S_{\mathcal{I}}$.

– $z_{\mathcal{I}}(S)$: performance of schedule S realized on $\mathcal{I}$, simply denoted $z_{\mathcal{I}}$ if there is no ambiguity.

– $z^*_{\mathcal{I}}$: performance of an optimal schedule on $\mathcal{I}$.

In deterministic scheduling, the performance criterion is fixed from the beginning, and it is immediately computable for a fixed schedule. In scheduling with uncertainties, there are several possible measures for a given criterion, and we try to give below a typology of these measures.

1.5.1. *Flexibility as a robustness indicator*

As said before, flexibility is the freedom allowed at execution phase for building the final schedule. Intuitively, it should be easier to propose a robust method if the allowed flexibility is large. Let us think about this. If everything is possible, we could be tempted to report any decision at execution phase (reactive approach); this is not always the best for the quality of the final solutions (myopic behavior, temporal constraints). Another key feature of a method is its feasibility for all considered uncertainties. If uncertainty is large, and/or disturbances numerous, the feasibility cannot be guaranteed in general (unless some exceptional repair mechanism can be set). Hence we should always try to maximize the method flexibility, expressed as its feasibility for the largest set of scenarios (here understood in a broad sense). It must be noted that starting from the internal or allowed flexibility, we consider now a chosen flexibility. In that sense, a flexibility indicator can rightly be considered as a robustness measure for the method at hand; see Chapters 9 and 11 in this book.

Still, it is always a good idea to couple that indicator with another measure, bound to the performance in the classical sense. Concerning the studies from the literature, the feasibility guarantee is usually implicitly accepted as a property of the method (a hypothesis that should be justified). In that case, flexibility is not measured.

1.5.2. *Schedule stability (solution robustness)*

Here the performance criterion (makespan, mean flow-time, etc.) is not considered. For a given method, we try to minimize the differences between the different solutions obtained (by the same method) for different scenarios. In automation literature this specific aspect of robustness is sometimes called *stability*, a term that shall be used in that sense inside this book (see Chapters 8 and 13); ideally, there is one schedule unchanged for the different scenarios, hence it is *stable*. The difference between two schedules, denoted their distance d, can be for instance the number of permutations between tasks or machines, or any other adequate measure in the considered context. Then we might try to minimize either the largest distance between two solutions, or the largest distance with respect to some reference, or baseline, schedule $\tilde{S}$. In the second case, we speak naturally of the stability of this baseline schedule, or of *solution robustness* [HER 05].

$$R_1 = \max_{\mathcal{I},\mathcal{I}' \in \mathcal{P}} d\big(S_{\mathcal{I}}, S'_{\mathcal{I}}\big)$$

$$R'_1 = \max_{\mathcal{I} \in \mathcal{P}} d\big(S_{\mathcal{I}}, \tilde{S}\big)$$

In a way, we try to build the smallest set of schedules compatible with the uncertainty taken into account. Equivalently, it is the search for a solution set of minimum flexibility, but sufficient to guarantee feasibility. In practice, we usually start from a reference (or baseline) schedule whose performances are acceptable for the available estimations.

1.5.3. *Stability relatively to a performance criterion (quality robustness)*

Again, we try to minimize a distance between the solutions obtained by different scenarios. This time though, the distance is measured with respect to the obtained value of the criterion. In [HER 05] this is called *quality robustness*, meaning that the quality (criterion value) of the baseline schedule should remain equivalent in all scenarios. With different models and points of view, such robustness measures are used in Chapters 6, 7, 12 and 14. If this value is considered as one characteristic, among others, of a schedule, it is a particular example of the previous case: we look for a set of solutions with close performances. Most often this approach is used jointly with a model based on estimations of the data (scenario $\tilde{\mathcal{I}}$). There is a baseline schedule, and the robustness measure is given by the largest difference between the performance of this schedule for the initial scenario $z_{\tilde{\mathcal{I}}}$, and the performance obtained for any other scenario:

$$R_2 = \max_{\mathcal{I}\in\mathcal{P}} \frac{z_{\mathcal{I}}}{z_{\tilde{\mathcal{I}}}} \qquad \text{(relative difference)}$$

$$R_2' = \max_{\mathcal{I}\in\mathcal{P}} \left|z_{\mathcal{I}} - z_{\tilde{\mathcal{I}}}\right| \qquad \text{(absolute difference)}$$

R_2 can be called the *stability ratio*. In one important case, only one schedule is built whatever the scenario (this implies the feasibility hypothesis). In fact, a family of schedules is usually considered, obtained from an initial schedule by accepting some amount of temporal flexibility. The robustness of this schedule S is measured by R_2 or R_2', $z_{\mathcal{I}}$ being here the performance of S for $\mathcal{I}$.

From the perspective of the associated optimization problems, such as looking for the most robust process for R_2, the problem is equivalent to minimizing the largest value of the criterion on the set of possible scenarios as soon as $z_{\tilde{\mathcal{I}}}$ is fixed.

When statistical data are available, a stochastic model can be used and it is possible to look for schedules whose mean behavior is good. Although the robustness measures are obtained from R_2 or R_2' by replacing the maximum by, for instance, the expectation, it is less natural to speak of stability (except that it means some guarantees can be obtained about, for instance, the mean behavior). The obtained measures are here the traditional measures in stochastic optimization, particularly:

$$R_3 = E_{\mathcal{I}\in\mathcal{P}}\left[z_{\mathcal{I}}\right] \quad \text{(criterion expectation)}$$

The drawback of all these measures is that they sometimes lead to schedules quite far from the best solution for a given scenario, even if it is reassuring to build a stable set of solution, or a set with good mean behavior. That is why many authors keep the term robustness for other measures.

1.5.4. *Respect of a fixed performance threshold*

Frequently, a target performance $\tilde{z}$ is defined *a priori*. The method is then considered as robust if no performance schedule exceeds this threshold, whatever the scenario. Hence the measure

$$R_4 = \max_{\mathcal{I}\in\mathcal{P}} \big(z_{\mathcal{I}} - \tilde{z}\big) \quad \text{(absolute deviation with respect to some threshold)}$$

Of course, the associated optimization problem is the same as for the stability with respect to the performance criterion (see section 1.5.3) and the same drawback holds.

In the case of a stochastic model, it is logical (see Daniels and Carillo [DAN 97]) to minimize the probability of exceeding the threshold:

$$R_5 = P_{\mathcal{I}\in\mathcal{P}}\big(z_{\mathcal{I}} \geq \tilde{z}\big) \quad \text{(service level measure)}$$

The difficulty in using this measure (see Chapter 5) comes from the fact that we must know the probability law associated with the random variable z.

1.5.5. *Deviation measures with respect to the optimum*

Robustness is measured here by comparing the criterion value obtained by the method and the optimum value, this for all scenarios. Following [DAN 95, KOU 97] we use the term of deviation: absolute deviation if the difference is computed; relative deviation for the ratio. There are several possibilities for using these deviations. One possibility is to compute their maxima, or their expectation in a stochastic setting. Thus, two relative measures are possible:

$$R_6 = \max_{\mathcal{I}\in\mathcal{P}} \frac{z_{\mathcal{I}}}{z^*_{\mathcal{I}}} \qquad \text{(worst case relative deviation)}$$

$$R_7 = E_{\mathcal{I}\in\mathcal{P}}\left[\frac{z_{\mathcal{I}}}{z^*_{\mathcal{I}}}\right] \qquad \text{(relative deviation in expectation)}$$

and the corresponding absolute measures R'_6 and R'_7. In robust optimization literature (see [AÏS 07, AVE 00, MUL 95]), the absolute deviation is called the "*minmax regret criterion*". Note also that in approximation literature, and sometimes in on-line optimization, the term *competitivity ratio* can be used for the relative deviation.

Computing these measures supposes that the optimal solution can be computed for each scenario, and in the stochastic case, that the probability of each scenario is known. This is not always possible, and we might just compute some upper bound, called the *sensitivity bound*, absolute or relative (see Chapter 4). It is also possible in the stochastic case to compute an approximation of R_7 or R'_7, by computing a mean after sampling the scenarios, see Kouvelis *et al.* [KOU 00] and more generally the stochastic optimization literature. We might speak then of *sampled mean deviation*.

1.5.6. *Sensitivity and robustness*

In the literature, it is sometimes difficult to separate sensitivity analysis and robustness. In fact the sensitivity analysis tries to answer the "what if..." questions. It deals with disturbances more than with general uncertainty: data are fixed but might be disturbed, a baseline schedule $\tilde{S}$ is given, which is most often optimal. Sensitivity analysis tries to measure the performance degradation of $\tilde{S}$ for a particular disturbance. It is not concerned with the execution phase: the robustness of a static algorithm is measured. Among the above measures, sensitivity analysis might use R'_1, R_2, R_6 or R'_6, and does not deal with probabilistic models. Furthermore, it comes historically from linear programming (LP), and as for LP disturbances concerns one parameter at a time. In LP, the maximum change of a parameter for which the current basis is still optimal is easy to compute. In scheduling, Sotskov *et al.* [SOT 98] did introduce the stability radius $\rho_{\tilde{S}}$. Computing the radius is equivalent to searching for the maximum disturbance size for which $R'_2 = 0$. Hall and Posner [HAL 04] present a classification and many results about sensitivity analysis in scheduling.

It is of course possible to extend the analysis to a true uncertainty on data simultaneously considered (see [PEN 01] and Chapters 3 and 4 in this book), if the study is restricted to the static phase and to some disturbance types (taking into account the breakdowns, for instance, seems impossible). However, it is then difficult to show results on the stability radius for instance.

1.6. Bibliography

[AÏS 07] AÏSSI H., BAZGAN C. and VANDERPOOTEN D., "Min-max and min-max regret versions of some combinatorial optimization problems: a survey", p. 1–32, Annales du Lamsade. ROY B., ALOULOU M.A. and KALAI R. (Eds.): *Robustness in OR-DA*, Paris, 2007.

[ALO 02] ALOULOU M.A., PORTMANN M.-C. and VIGNIER A., "Predictive-reactive scheduling for the single machine problem", *Proceedings of the 8th International Workshop on Project Management and Scheduling*, Valencia, Spain, p. 39–42, 2002.

[ART 99] ARTIGUES C., ROUBELLAT F. and BILLAUT J.-C., “Characterization of a set of schedules in a resource-constrained multi-project scheduling problem with multiple modes”, *International Journal of Industrial Engineering*, vol. 6, no. 2, p. 112–122, 1999.

[ART 05] ARTIGUES C., BILLAUT J.-C. and ESSWEIN C., “Maximization of solution flexibility for robust shop scheduling”, *European Journal of Operational Research*, vol. 165, no. 2, p. 314–328, 2005.

[AVE 00] AVERBAKH I., “On the complexity of a class of combinatorial optimization problems with uncertainty”, *Mathematical Programming*, vol. Ser A 90, p. 263–272, 2000.

[BEN 99] BEN-TAL A. and NEMIROVSKI A., “Robust solutions of uncertain linear programs”, *Operations Research Letters*, vol. 25, p. 1–13, 1999.

[BLA 01] BLAZEWICZ J., ECKER K.H., PESCH E., SCHMIDT G. and WEGLARZ J., *Scheduling Computer and Manufacturing Processes*, Springer-Verlag, Berlin, 2nd edition, 2001.

[BRU 07] BRUCKER P., *Scheduling Algorithms*, Springer-Verlag, Berlin, 5th edition, 2007.

[CHR 95] CHRÉTIENNE P., COFFMAN JR. E.G., LENSTRA J.K. and LIU Z. (Eds.), *Scheduling Theory and its Applications*, John Wiley & Sons, 1995.

[DAN 95] DANIELS R.L. and KOUVELIS P., “Robust scheduling to hedge against processing time uncertainty in single stage production”, *Management Science*, vol. 41, no. 2, p. 363–376, 1995.

[DAN 97] DANIELS R.L. and CARILLO J.E., “β-robust scheduling for single-machine systems with uncertain processing times”, *IIE Transactions*, vol. 29, p. 977–985, 1997.

[DAV 00] DAVENPORT A.J. and BECK J.C., A survey of techniques for scheduling with uncertainty, available in http://www.eil.utoronto.ca/profiles/chris/chris.papers.html, 2000.

[DUB 03] DUBOIS D., FARGIER H. and FORTEMPS B., “Fuzzy scheduling: modelling flexible constraints vs coping with incomplete knowledge”, *European Journal of Operational Research*, vol. 147, p. 231–252, 2003.

[FIA 98] FIAT A. and WOEGINGER G.J. (Eds.), *Online Algorithms, The State of the Art*, Lecture Notes in Computer Science, Springer, 1998.

[GOL 97] GOLDRATT E.M., *Critical Chain*, The North River Press Publishing Corporation, Great Barrington, 1997.

[GRA 79] GRAHAM R.E., LAWLER E.L., LENSTRA J.K. and RINNOOY KAN A., “Optimization and approximation in deterministic sequencing and scheduling: a survey”, *Ann. Discrete Math.*, vol. 4, p. 287–326, 1979.

[HAL 04] HALL N. and POSNER M., “Sensitivity analysis for scheduling problems”, *Journal of Scheduling*, vol. 7, p. 49–83, 2004.