CAM

Control Systems, Robotics and Manufacturing Series

Production Scheduling

Edited by Pierre Lopez and François Roubellat

ISTE

WILEY

This page intentionally left blank

Production Scheduling

This page intentionally left blank

Production Scheduling

Edited by
Pierre Lopez
François Roubellat

ISTE
WILEY

First published in France in 2001 by Hermes Science entitled “Ordonnancement de la production”
First published in Great Britain and the United States in 2008 by ISTE Ltd and John Wiley & Sons, Inc.

ISTE Ltd
6 Fitzroy Square
London W1T 5DX
UK

www.iste.co.uk

John Wiley & Sons, Inc.
111 River Street
Hoboken, NJ 07030
USA

www.wiley.com

Library of Congress Cataloging-in-Publication Data

[Ordonnancement de la production. English]
Production scheduling / edited by Pierre Lopez, Francois Roubellat.
p. cm.
Includes bibliographical references and index.
ISBN 978-1-84821-017-2
1. Production scheduling. 2. Inventory control. I. Lopez, Pierre. II. Roubellat, Francois. III. Title.
TS157.5.07313 2008
658.5'3--dc22

2007043950

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN: 978-1-84821-017-2

Printed and bound in Great Britain by Antony Rowe Ltd, Chippenham, Wiltshire.

Table of Contents

Preface

This book is mainly a translation of an earlier text published in French in 2001. For this edition, all the authors have been requested to update their chapter by considering the latest advances and references in the field.

Research studies on job scheduling are mobilizing a large number of researchers attempting to bring solutions to theoretical as well as practical problems. New resolution methods are being developed in relation to classical scheduling problems or practical situations encountered in the context of goods or service production. Exchanges between researchers in this community are extensive and contribute to a real dynamic in this field. Numerous research studies are developed in cooperation with companies to identify real problems including problem statements not considered in much basic research for scheduling problem solving.

In this book, we have attempted to bring together a number of contributions liable to help in resolving or considering a solution to problems actually encountered in the real world.

We wish to thank all the authors who have graciously accepted to participate and who have presented a summary of available results or current studies in different fields related to production scheduling.

Pierre LOPEZ
François ROUBELLAT

This page intentionally left blank

Chapter 1

Statement of Production Scheduling

The current environment in companies is characterized by markets facing fierce competition and from which customer requirements and expectations are becoming increasingly high in terms of quality, cost and delivery times. This evolution is made even stronger by rapid development of new information and communication technologies which provide a direct connection between companies (business to business) and between companies and their clients (business to customer). In this type of context, company performance is built on two dimensions:

– a technological dimension, whose goal is to develop intrinsic performance of marketed products in order to satisfy requirements of quality and lower cost of ownership for these products. Technological innovation plays an important role and can be a differentiating element for market development and penetration. In this regard, we must note that rapid product technological growth and the personalization requirements for these products expected by markets often lead companies to forsake mass production and instead focus on small or medium-sized production runs, even on-demand manufacturing. This requires them to have flexible and progressive production systems, able to adapt to market demands and needs quickly and efficiently;

– an organizational dimension intended for performance development in terms of production cycle times, respect of expected delivery dates, inventory and work in process management, adaptation and reactivity to variations in commercial orders, etc. This dimension plays an increasingly important role as markets are increasingly volatile and progressive, and require shorter response times from companies.

Chapter written by François ROUBELLAT and Pierre LOPEZ.

Therefore, companies must have powerful methods and tools at their disposal for production organization and control [BAI 99].

This production organization must be considered not only at company level, but also from its position in the supply chain where it is one of the links, resulting in a global "virtual" company which must be focused on satisfying customer needs under the best possible conditions [BAI 99, WU 98].

To achieve these goals, a company organization normally relies on the implementation of a number of functions including scheduling which plays a vital role. Indeed, the scheduling function is intended for the organization of human and technological resource use in company workshops to directly satisfy client requirements or demands issued from a production plan prepared by the company planning function. Considering market trends and requirements, this function must organize the simultaneous execution of several jobs using flexible resources available in limited amounts, which becomes a complex problem to solve. In addition, it is this function which ultimately is responsible for product manufacturing. Its efficiency and failures will therefore highly condition the company's relationship with its customers. Within companies, this function has obviously always been present, but today it must face increasingly complex problems because of the large number of jobs that must be executed simultaneously with shorter manufacturing times. This situation is obviously the result of the current environment as it was described earlier.

Offering efficient and powerful solutions to scheduling problems thus defined constitutes an important economic challenge. Despite the simplicity of formulating this type of problem, it must be noted that to date there is no "one" method able to solve all possible scenarios. In fact there are a number of generic problems differentiated by the characteristics of jobs to be performed or resources available to perform them. Specific methods can then be associated with the resolution of each of these generic problems, these specific methods being either a specific interpretation of the problem in a general way, or a specific method dedicated to the problem involved. Resolution of a concrete problem starts by identifying the generic problem to which we can associate it, followed by the selection of the method(s) adapted to the resolution of this problem. We must also note that the decision problem associated with the scheduling problem belongs to the category of combinatorial NP-complete problems. Consequently, resolution by exact methods is not realistic for large problems, justifying the use of powerful heuristic methods. This explains why research on scheduling problems is always popular and demands numerous studies [WIE 97].

The goal of this book is to present a number of methods for the resolution of scheduling problems. There are a number of studies, past or present, on scheduling

issues [HER 06, LEU 04, PIN 05, SUL 07]. In this book, we have chosen to emphasize approaches to strictly solve or to improve the solution for problems actually encountered in the real world. This leads us to the organization of this book as follows.

A global presentation of concepts in relation to the scheduling domain and basic methods for solving classical scheduling problems is proposed in Chapter 2. The following two chapters present heuristic type resolution methods to solve large problems: metaheuristics (Chapter 3) and genetic algorithms (Chapter 4). Chapter 5 discusses an approach based on constraint propagation aiming at characterizing all solutions to a problem, which can be useful to help in searching for optimal solutions or in decision support. Chapter 6 recalls resolution principles based on the use of priority rules to generate a schedule by simulation or to identify priority rules to be used to organize scheduling in real time. The two following chapters discuss three scheduling problems for specific production lines: cyclic scheduling to satisfy mass production of a limited number of products (Chapter 7) and hoist scheduling (Chapter 8). Chapter 9 addresses an important practical extension of the scheduling problem: choosing which resource to use for each job operation, coupled with the operation scheduling for resources used. Chapter 10 considers a specific problem for which operations to be performed are not linked to sequencing constraints. Finally, the last two chapters focus on scheduling problems in uncertain environments or in environments which are not completely specified, a situation often encountered in practice. The fuzzy approach makes it possible to consider these two characteristics (Chapter 11), whereas the decision support approach makes it possible to organize job processing in real time according to the real state of the workshop and thus to consider unforeseen situations, using deterministic models (Chapter 12).

It is important to note that although this book often uses vocabulary and examples inherent to the manufacturing field, methods presented can be used to organize processing of any type of activity requiring available resources in a limited amount, in particular service activities, provided of course that the processing constraints are similar to those of one of the models considered in this book.

Bibliography

[BAI 99] BAI S., NSF workshop on supply chain management in electronic commerce, Final Report, http://www.ise.ufl.edu/supplychain, 1998.

[HER 06] HERRMANN J. (Ed.), *Handbook of Production Scheduling*, International Series in Operations Research & Management Science , vol. 89, 2006.

[LEU 04] LEUNG J.Y-T., *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Chapman & Hall/CRC Computer & Information Science Series, 2004.

[PIN 05] PINEDO M., *Planning and Scheduling in Manufacturing and Services*, Springer, 2005.

[SUL 07] SULE D., *Final Production Planning and Industrial Scheduling: Examples, Case Studies and Applications*, Second Edition, Taylor & Francis/CRC Press, 2007.

[WIE 97] WIERS V.C.S., "A review of the applicability of OR and AI scheduling techniques in practice", *Omega, International Journal of Management Science*, vol. 25, no. 2, p. 145-153, 1997.

[WU 98] WU S.D., STORER R.H. and MARTIN-VEGA L.A., "Manufacturing logistics research: taxonomy and directions", *NSF Design and Manufacturing Grantees Conference*, Monterrey, Mexico, 1995.

Chapter 2

Basic Concepts and Methods in Production Scheduling

2.1. Introduction

The scheduling problem is the organization over time of the execution of a set of tasks, taking into account time constraints (i.e. deadlines, precedence constraints, etc.) and capability and capacity constraints on resources required for these tasks.

A schedule constitutes a solution to the scheduling problem. It describes the execution of tasks and the allocation of resources over time with the aim of meeting one or more objectives. More precisely scheduling problems can be decomposed in two types. In a *pure scheduling problem* a start date and an end date have to be decided for each task, whereas in a *sequencing problem* tasks that compete for the use of the same resource only have to be ordered. Scheduling necessarily induces a unique set of sequencing relations. On the other hand, a solution to the sequencing problem, described in terms of precedence relations (those that solve the potential resources conflicts) is not associated with a single schedule but covers a family of schedules (possibly an infinity of schedules if dates are real numbers or if the variation domain is not bounded).

This chapter aims to present the main results from basic literature on scheduling problems, results which may be referred to in the following chapters. First, we present the basic concepts by integrating general method principles for the resolution of optimization problems. Then we address the project scheduling

---

Chapter written by Patrick ESQUIROL and Pierre LOPEZ.

problem where most questions involve the determination of total project time and the emphasis of usable degrees of freedom (time floats) to minimize the cost of resource utilization or to smooth out the workload. Although this type of problem does not take resource constraints into account, its formulation is essential in scheduling because it enables the presentation of fundamental concepts and basic algorithms for handling time constraints. Finally, we present a second important family of problems, the shop scheduling problems. While in project scheduling, we can adapt the level of resources necessary for the execution of complex but well structured work, in shop scheduling on the other hand, we must use a known set of limited resources (machines) as well as possible to produce a diverse set of products. The complexity then lies in the combinatorics resulting from the consideration of existing resources limitation and not in the manufacturing processes which are here predetermined (in shop scheduling, manufacturing processes are often restricted to be linear sequences of operations on distinct machines, called routings). A series of traditional results are presented concerning initially one machine problems, then parallel machine problems and finally problems where products circulate in the shop according to a single routing or multiple routings.

2.2. Basic scheduling concepts

2.2.1. *Tasks*

A *task* is a basic work entity located in time with a start time t_i and/or an end time c_i, where execution is characterized by a *duration* p_i (we have $c_i = t_i + p_i$) and by the *intensity* a_i^k with which it consumes certain *resources* k. In more simple terms, we presume that for each resource required, this intensity is constant during task execution.

With certain problems, the tasks can be executed in parts, and interlacing of the different parts ensures that resources are as active as possible. With others, on the other hand, we cannot stop a task that is started. We speak of *preemptive* and *non-preemptive* problems respectively.

In project scheduling (see section 2.3), we will keep the term “task” to describe the activities of a project. For shop scheduling (see section 2.4), we will opt for the term “operation” (in this case, we will also prefer *processing time* rather than duration). In manufacturing, we often observe several phases in the execution of an operation: preparation; main phase; finishing; and transport. Depending on manufacturing conditions, each of these phases can greatly condition the time in which a product remains in the shop.

2.2.2. *Resources*

A resource k is a technical or human means needed for the execution of a task and available in a limited quantity, its (presumed constant) *capacity* A_k. Several types of resources can be observed. A resource is said to be *renewable* if it becomes available again with the same quantity (man, machine, space, equipment in general, etc.) once used by one or more tasks; usable resource quantity is limited at each moment. Otherwise, it is *non-renewable* or *consumable* (raw material, budget, etc.); global consumption (or accumulation) during time is limited. A resource is said to be *doubly-constrained* when its instant use and global consumption are both limited (source of energy, financing, etc.). *Disjunctive* resources cannot be shared, they are able to execute only one task at a time (machine tool, manipulating robot), while *cumulative* resources can be shared by several tasks simultaneously (team of workers, workstation) assuming that their capacity is sufficient.

NOTE.– In practice, flexibility of certain resources can lead to taking into account additional reconfiguration times, which may depend on the task at hand, but also on the state of the resource itself, which is often linked to the previous tasks executed on this resource. This means that setup times must be taken into consideration, depending on the sequence chosen. The consideration of these setup times is vital in practice because they can have a huge impact on scheduling. It encourages the grouping of identical products into lots resulting in grouping tasks relative to a single lot on each machine in order to minimize total machine reconfiguration time (see Chapter 12). In addition, task durations are not always known but can be dependent on the quantity of means alloted to their execution which in turn can be linked to the resource's speed or performance (see Chapters 5, 9 and 12).

2.2.3. *Modeling*

Generally, the decision variables encountered in this book involve decisions on time (scheduling variables) or resources (allocation variables). A constraint expresses restrictions on the values that decision variables can jointly take. Time and resource constraints can be distinguished.

2.2.3.1. *Time constraints*

These include:

– allocated time constraints, generally based on a management policy and relative to task deadlines (delivery lead times, availability of supply) or to project deadlines. For a task i, r_i defines the date of availability (before which i cannot begin) and d_i defines its due date (before which i must be completed);

– precedence constraints and, in a more general way, technological consistency constraints, which describe a relative ordering of certain tasks compared to others (for example, routing constraints in the case of shop problems);

– timetable constraints linked to respect for work schedules, etc.

These constraints can all be expressed with the help of *potential inequalities* [ROY 70], which mandate a minimum distance of b_{ij} between two particular events associated with tasks (their starting times or their ending times):

$$x_j - x_i \geq b_{ij} \qquad [2.1]$$

Now that this primitive constraint is defined, we will examine how it can be used to express some of the time constraints presented above.

2.2.3.1.1. Constraint expressed in a single potential inequality

The precedence constraint between two tasks i and j, symbolized as $i \prec j$ (i precedes j), is represented by the numerical relation $t_j - t_i \geq p_i$.

A time limit constraint can also be represented with the help of the primitive constraint [2.1], by including a dummy task 0 with a zero duration, that sets the time origin ($t_0 = 0$) of the problem; for example, constraint $t_i \geq r_i$ is rewritten in the form of $t_i - t_0 \geq r_i$. The constraint $c_i \leq d_i$ becomes $t_0 - c_i \geq -d_i$.

2.2.3.1.2. Constraint expressed with a conjunctive set of potential inequalities

This is the case with relative location constraints such as a necessary overlap between two tasks i and j. It is represented by a constraint requiring two potential inequalities:

$$(c_j - t_i \geq 0) \wedge (c_i - t_j \geq 0).$$

We use this type of *conjunctive* logical expression to represent that a time constraint is satisfied if all its member constraints are simultaneously satisfied: connector $\wedge$ represents the Boolean *logical and* connector, potential inequalities are then interpreted as logical literals of the first order predicate calculus.

Another example of a complex time constraint resulting in a conjunctive expression is that of tasks with a constrained duration. To model the fact that the duration of a task i is only approximately known and defined by an interval of possible values of the type $[\underline{p}_i, \overline{p}_i]$, the following conjunctive expression can be used:

$$(c_i - t_i \geq \underline{p}_i) \wedge (t_i - c_i \geq -\overline{p}_i).$$

2.2.3.1.3. Constraint expressed as a disjunction of potential inequalities

This type of expression involves the connector $\vee$ (*or* logical). Semantically, the constraint is satisfied if at least one of the literals is a satisfied constraint.

A first case is for example the existence of several execution modes. For example, let us define a product which can be manufactured in two distinct ways:

– mode 1: operation *i*, then operation *j*;

– mode 2: operation *j*, minimum waiting time of p_{wait} units (for cooling down, drying process, etc.), then operation *i*.

The global time constraint which expresses the existence of these two execution modes can have the following form:

$$(t_j - c_i \geq 0) \vee (t_i - c_j \geq p_{wait}) .$$

In the simple previous case, the time constraint is directly represented in the form of a disjunction between several potential inequalities. A more complex case is the one of timetable constraints: task i can be accomplished only during one of some h time intervals: $[r_i^1, d_i^1]$, $[r_i^2, d_i^2]$, …, $[r_i^h, d_i^h]$, with $[r_i^l, d_i^l] \cap [r_i^{l+1}, d_i^{l+1}] = \varnothing$, $\forall l = 1, \ldots, (h-1)$. If there is no hypothesis on the timetable for i, the constraint is represented with the help of the following disjunctive expression:

$$[(t_i - t_0 \geq r_i^1) \wedge (t_0 - c_i \geq -d_i^1)] \vee [(t_i - t_0 \geq r_i^2) \wedge (t_0 - c_i \geq -d_i^2)] \vee \ldots$$
$$\ldots \vee [(t_i - t_0 \geq r_i^h) \wedge (t_0 - c_i \geq -d_i^h)]$$

This form is not a disjunctive set of single potentials inequalities. However, thanks to a rewriting using the distributive nature of logical connectors, we can translate this expression into a conjunctive set of disjunctive expressions on potential inequalities, i.e. into a set of constraints to satisfy, where each one is a disjunctive expression.

2.2.3.2. *Resource constraints and sequencing problem*

These constraints signify that resources are available in limited quantity (their capacity), and are also called sharing constraints. For a given schedule, let the set of tasks consuming resource k at time t be: $T_k(t) = \{i \in 1,\ldots,n \mid t \in [t_i, t_i + p_i[\}$ (note that tasks ending exactly at moment t, are not considered in $T_k(t)$). Capacity limitation of resource k is then expressed by the following constraint:

$$\forall t, \sum_{i \in T_k(t)} a_i \leq A_k \qquad [2.2]$$

To ensure respect of resource constraints, we must avoid temporal overlapping of certain task subsets. This leads to the order of a sufficient number of task pairs, which comes down to adding new precedence constraints to the initial constraint set. This type of decision is called a *sequencing decision* or an *arbitration*. All sequencing decisions characterize the *sequencing problem* brought about by the initial scheduling problem. There are two types of resource constraints, linked to the disjunctive or cumulative nature of resources.

Disjunctive resources

These are resources which can only be used for one task at a time. In a shop problem, resources are machines. A *disjunction pair* $(i \prec j) \vee (j \prec i)$ is associated with each pair (i, j) of tasks using the same resource, which is translated in the form of a disjunction between two potential inequalities:

$$(t_j - t_i \geq p_i) \vee (t_i - t_j \geq p_j) \qquad [2.3]$$

In order to respect these constraints, it is necessary and sufficient to solve all disjunction pairs relative to each disjunctive resource, which leads to a *total sequencing* of tasks using a single disjunctive resource.

Cumulative resources

When the resource capacity and task intensities are greater than one, the set of tasks which cannot be executed simultaneously may have a higher cardinality. The respect of resource constraints only leads to a partial sequence of tasks using the same resource. As before, it is possible to represent cumulative resource constraints with the help of a conjunction of disjunctive expressions. Chapters 5, 9 and 12 describe the involvement of cumulative resources.

2.2.3.3. *Objectives and evaluation criteria*

When we address the resolution of a scheduling problem, we can choose between two main strategy types, respectively focussing on the optimality of solutions (in relation to one or more criteria), or on their feasibility (in relation to constraints). The optimization-based approach presumes that candidate solutions for a problem can be ordered, according to one or more numerical evaluation criteria that express the quality of solutions. We will attempt to minimize or maximize such criteria. Note for example those:

– related to time: total execution time or average time for completing a set of tasks; the different types of delays in relation to deadlines set;

– related to resources: the – maximum, average or weighted – quantity of resources necessary for completing a set of tasks and the load of each resource;

– related to costs for launching, production, transportation, storage, etc.

A criterion based on variables c_i is called *regular* if we cannot downgrade it as a task is executed earlier. This is the case for minimization of makespan, maximum delay, etc. A subset of solutions is called *dominant* for the optimization of a given criterion if it contains at least one *optimum* for this criterion. In this way, the search for an optimal solution can be limited to a dominant subset. In a *semi-active* schedule, we cannot move a task earlier without modifying the sequence on the resource it uses. The set of semi-active schedules is dominant for any regular criterion. In an *active* schedule, no task can be started earlier without delaying the start of another one. Active schedules are semi-active and the set of active schedules is dominant for any regular criterion. In *non-delay* schedules, we must not delay the execution of a task if it is ready to start and if its resource is available. Non-delay schedules are also active. We will come back to the generation of these schedules in section 2.4.6 and Chapter 6.

It is sometimes difficult to translate all solving objectives with one or more numerical criteria. In this case, we can use a *constraint satisfaction* approach. The set of constraints groups intrinsic problem constraints (technological consistency for example) as well as threshold objectives to be reached or not to be exceeded (maximum duration, maximum/minimum inventory, etc.). In this case, we will be able to use an ordinary solution as long as it is feasible. The exploration strategy of the space of feasible solutions must use constraints in a smart way in order to reduce the space exploration. This is the principle of constraint propagation, representing a set of filtering techniques (withdrawal of inconsistent values) and developed in Chapter 5.

2.2.4. *Resolution methods*

2.2.4.1. *Introduction*

In an optimization-based approach, the knowledge helping to provide an ideal solution is integrated in a programmable model where the execution does not require the intervention of a decision maker. When, on the contrary, some knowledge is difficult to represent or use (uncertain, inaccurate data, non-consistent, qualitative or highly context-dependent criteria), an approach based on decision support may turn out to be more realistic and flexible. In this case, the idea is to analyze the problem and characterize all solutions in order to help the decision maker with total or partial control of the resolution procedure in solving the problem. For this, characterization can rely on constraint propagation techniques.

A method using an optimization criterion is *exact* if it guarantees optimality of solutions found. Otherwise it will be called approximate, or *heuristic* when it empirically provides "good" solutions [MAC 93]. If the goal is reduced to only satisfy constraints, a method based on constraint propagation will be said to be exact (or "sound") if the solutions found correctly satisfy all constraints of the problem. Finally, we should mention the completeness property; a method is complete if, regardless of the problem addressed, we can ensure that the problem accepts or does not accept solutions in a reasonable amount of time.

Calculation cost is clearly vital when choosing an automatic resolution method. Even though the increasing power of computers may constantly push back the limit of combinatorial problem sizes accessible by exact methods, their resolution is often expensive (computing time, memory space), which explains the use of approximate methods. In the case of an interactive resolution, the number of decisions made by the decision maker, the number of backtracks, …, are also factors conditioning this cost independently from that incurred by underlying techniques used.

2.2.4.2. *General methods of combinatorial optimization*

We briefly describe here a few general optimization methods that we will often encounter in the exact resolution of scheduling problems: branch-and-bound procedures [BAK 74] and mathematical programming techniques such as mixed-integer linear programming [NEM 88] and dynamic programming [BER 87].

2.2.4.2.1. Branch and bound procedures

Branch and bound (B&B) optimization procedures are exploration procedures using implicit enumeration of all solutions [LAW 66]. They lead to the development of a search tree where each node represents a sub-problem and where arcs issued from a single node represent all possible decompositions of the problem located at

the base of the arc into smaller size sub-problems (Figure 2.1). The objective is to arrive at a node corresponding to a solution, by generating the least number of nodes.

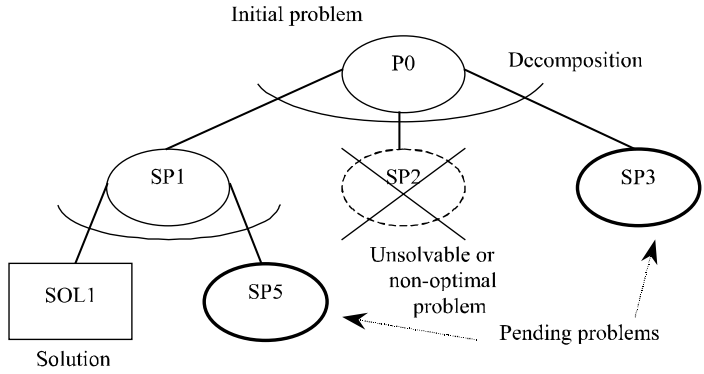


Figure 2.1. *Research space associated with a B&B procedure*

B&B procedures are based on four main components:

– the *branch* technique for decomposing a problem by breaking it down into smaller sized sub-problems;

– the *bound* method which involves a bound on the optimization criterion for all solutions of a sub-problem (lower bound for minimization);

– the *evaluation* method, for determining if a node is terminal (it contains no admissible solution, or it is an optimal solution, or we can obtain the optimal solution for this sub-problem in a polynomial time[1]), or if it should be separated. If we can make sure that it does not contain a better solution than the ones already found, it will not be separated;

– the *selection* method, or exploration strategy, which describes how to choose the sub-problem to branch, when there are several candidates. Two main strategies exist. In a best-first strategy, we select and then branch the "pending" sub-problem with the best bound. In a depth-first search procedure with backtrack, pending nodes are managed in a stack: at each step we seek a pending problem, we bind it, we probe it and we stack the sub-problems emanating from its branching.

1 A *polynomial* or *polynomial-time* solving method is a method with a running time bounded by a polynomial function of the problem size N (e.g. N^2). A problem is said to be easy if such methods exist. On the other hand, facing *hard combinatorial* problems, the running time is only bounded by an exponential function of N (e.g. 2^N).

For large problems, we must be concerned with having to handle substantial tree structures. To limit the search tree size, we can attempt to refine lower bound calculations. We can also apply dominance results [CAR 88, ESQ 99]. In addition to the lower bound calculation, we can calculate at each node an upper bound of the criterion with the help of a heuristic. The goal is to stop the development of a node when its lower bound is greater than the best upper bound found to date. Chapter 9 illustrates these methods.

2.2.4.2.2. Dynamic programming

Scheduling problems are combinatorial problems for which there are general tools such as dynamic programming [BEL 74] as long as this optimization criterion presents specific properties, such as an additive form. The principle is to carry out a step decomposition of the problem, and to scan backwards – from the last decision up – the sequential decision process associated with the scheduling problem. Each step corresponds to a sub-problem that we solve in an optimal way by considering information obtained during previous steps. This requires a formulation of the criterion in the form of a recurrence relation connecting two consecutive levels. As with B&B procedures, dynamic programming works by an implicit enumeration of all solutions. This optimization has a more general vocation than B&B procedures, but on the other hand, the size of problems that it can address is more limited. However, it is possible to apply dominance results, and for reasonably-sized NP-hard problems[2], we can, in practice, build interesting pseudo-polynomial[3] dynamic programming algorithms.

2.2.4.2.3. Linear and integer programming

A linear program models an optimization problem in which the criterion and constraints are linear functions of variables. To process a linear program in continuous variables, the two most important types of algorithms are the Simplex method and the Interior Points method. In practice, the Simplex method is powerful although its theoretical complexity is exponential. The best example of an Interior

---

2 In complexity theory [GAR 79], an optimization problem X is said to be NP-hard if the associated problem of existence E_X (finding an admissible solution at a cost lower than a given k) is an NP-complete problem. To do this, it must be proven that E_X accepts a formulation making it possible to easily verify (in polynomial time) whether a given set of values actually does constitute a solution (definition of the NP class). It must also be proven that we can find E_X by polynomial transformation of another known NP-complete problem (for example, the SAT problem raised by satisfiability of a set of Boolean formulae). In this way, finding a polynomial resolution algorithm for a single NP-complete problem would automatically solve all problems. This type of algorithm does not yet exist.

3 A pseudo-polynomial algorithm is an algorithm capable of solving an NP-complete problem in polynomial time with coding of the problem's data in base 1 (see [CAR 88], page 76). A problem for which no pseudo-polynomial algorithm exists will be said to be *NP-complete in the strong sense*.

Points method is the Karmarkar algorithm which solves linear programs in polynomial time.

Modeling scheduling problems often involves integer variables in the mathematical program. We are then in the presence of mixed-variable programs where the major disadvantage may be the large number of required constraints and variables; no polynomial algorithm exists to solve this type of program. The best approaches used relax the constraints that state integer values for variables; these are polyhedral methods derived from the Simplex algorithm, and B&B procedures. In certain cases, we can go back to *flow* problems for which we have powerful algorithms [WOL 98].

2.2.5. *Representation of solutions*

The *Gantt chart* is a very simple and widely used graphical representation for viewing scheduling. A horizontal segment of length that is proportional to the operation duration is associated with each task (Figure 2.2). In a "resource" chart, each horizontal line corresponds to a resource, which makes it possible to view its periods of operation or idleness as well as the sequence of operations using it and the scheduling duration. In a "job" or "product" chart, a line is associated with each job; when it follows a linear route it is then easy to see the chaining of its operations and the waiting time between two consecutive operations.

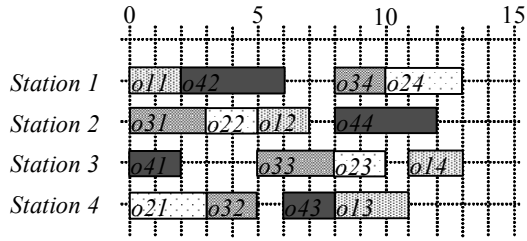


Figure 2.2. *"Resource" Gantt chart*

2.3. Project scheduling

This section addresses the description of a basic project scheduling method. This method is described in this book, even though it is focused on production scheduling, because it is a basic technique for methods developed in the context of production shops. The specific characteristic of project scheduling is to study a single project, for which we attempt to minimize makespan with no consideration for resource limitation constraints. Initially limited to managing large projects, the

use of these techniques, often grouped under the famous *PERT* acronym, has now spread to companies.

2.3.1. *Modeling*

Graph theory is a very real help in the manipulation of a large amount of numerical data. It is a rigorous support for both checking the consistency of the problem raised and its resolution. Current formulations linked to the project scheduling problem involve the definition of a graph using potential inequalities (see section 2.2.3).

When formulating potential inequalities on an *Activity-On-Node* graph, each node represents a task *i*, and each arc *(i, j)* represents a potential inequality between the start dates t_i and t_j of tasks *i* and *j* [ROY 70]. We generally add two nodes, *Beginning* and *End*, corresponding to fictitious project tasks with a zero time duration. In this way, we associate a zero value arc *(Beginning, i)* with each task *i* able to start the project; similarly we associate a zero value arc *(j, End)* with each task *j* able to complete the project. Subsequently, we use the traditional notation $\Gamma^-(i)$ (resp. $\Gamma^+(i)$) to represent the group of nodes located at the origin (resp. at the end) of arcs entering (resp. issued from) node *i*.

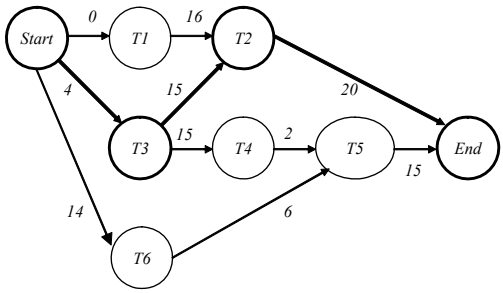


Figure 2.3. *Activity-On-Node graph associated with a project*

The above example (Figure 2.3) illustrates the case of a project containing six tasks submitted to different types of constraints (the durations are expressed in days):

– arcs $(T3, T2)$ and $(T3, T4)$ represent for example a precedence constraint and are valued by task $T3$ duration (15 days);

– arc *(Beginning,* $T1$*)* with value 0 indicates that task $T1$ can start when the project starts whereas task $T6$ must wait at least 14 days.

2.3.2. *Resolution*

One of the main questions in the resolution involves the calculation of the minimal project duration. This comes down to searching the longest path (arcs in bold in Figure 2.3) between *Beginning* and *End* nodes on the activity-on-node graph associated with the problem for which we have powerful algorithms. We thus determine *critical paths* each corresponding to a sequence of tasks where beginning dates are imposed if we wish to finish the project as soon as possible. They are called *critical path methods*, the most famous of which are PERT (program evaluation and review technique), CPM (critical path method) and MPM (Metra-potential method and, later, the project management method), all of which appeared in the late 1950s [ROY 70].

2.3.2.1. *Left-shift scheduling*

Calculating left-shift scheduling consists of allocating an earliest start time $\underline{t}_j$ to each task j, considering the earliest start times $\underline{t}_i$ of tasks i such that a $t_j - t_i \geq b_{ij}$ type constraint exists. Formally, the definition of earliest start times is recursive:

$$\underline{t}_j = \max_{i \in \Gamma^-(j)} \left(\underline{t}_i + b_{ij} \right) \qquad [2.4]$$

with: $\underline{t}_{Beginning} = 0$

nb_iter ← 0 *% Initialization*
for each node *j*, **do**
  $\underline{t}_j \leftarrow 0$
repeat *% Main loop*
  stable ← *true*
  for each node $j \neq Beginning$
    for any node $i \in \Gamma^-(j)$
      if $\underline{t}_j - \underline{t}_i < b_{ij}$ **then**
        $\underline{t}_j \leftarrow \underline{t}_i + b_{ij}$
        stable ← *false*
  nb_iter ← *nb_iter* + 1
until *stable* = *true* **or** *nb_iter* > *n* - 1
if *nb_iter* > *n* – 1 **then** *graph contains a circuit*

Algorithm 2.1. *Bellman algorithm*

Left-shift scheduling calculation determines the length of the longest paths reaching each task from the *Beginning* task. A solution exists if and only if no circuit of strictly positive length ("positive circuit") exists. The Bellman algorithm accomplishes this type of calculation. It ends in the presence of positive circuits, with the help of a theorem ensuring that for n nodes, the longest paths contain at the most $n-1$ arcs. Its statement is Algorithm 2.1.

NOTE.– When we can ensure the absence of circuits, the graph can be decomposed into levels and it is thus more interesting to apply a simplified algorithm traversing the nodes in the increasing order of levels (see for example [PRI 94]).

The application of Algorithm 2.1 makes it possible to determine the earliest finish time $\underline{t}_{End}$, and thus the minimum duration of the project.

2.3.2.2. *Right-shift scheduling*

If the objective is to end the project as soon as possible, we can rule that $\underline{t}_{End}$ is also a latest finish time, in order to determine by propagation a latest start time $\bar{t}_i$ for each task. This time, we calculate the length of the longest paths between any node and *end* node. To do this, we use a simplified dual procedure of the previous (in fact detection of positive length circuits is well established). Starting from an initial state where $\overline{t}_i \quad \overline{t}_{End} \quad \underline{t}_{End}$, the procedure provides a stable set of latest start times in at most $n-1$ iterations, confirming:

$$\bar{t}_i = \min_{j \in \Gamma^+(i)} \left(\bar{t}_j \quad b_{ij} \right) \qquad [2.5]$$

We call *critical tasks* the tasks which have identical earliest and latest start times once calculation of the left-shift scheduling and right-shift scheduling set at $\overline{t}_{End}$ has been found. By definition, *critical paths* connect the beginning and end of a project and are made up exclusively of critical tasks[4]. By determining critical paths, it then becomes possible to focus our attention on the execution of critical tasks. Any delay in the execution of a critical task inevitably prolongs the duration of the project.

For the others, we can determine several types of floats, which can possibly be used to smooth out workload, reduce costs, etc.

4 Note that it is not really necessary to calculate earliest and latest schedules to identify critical jobs and paths; when calculating left-shift scheduling, tasks involved in the longest paths need to be located.

2.3.2.3. *Characterization of the different float types*

Total float of a task is the maximum delay that it can have in relation to its earliest start time without affecting the minimum duration of the project. It is equal to the gap separating the earliest and latest start times:

$$TFloat_i = \bar{t}_i - \underline{t}_i$$

Free float of a task is the maximum delay that it can have in relation to its earliest start time without affecting its following tasks which will retain their same earliest start times:

$$FFloat_i = \min_{j \in \Gamma^+(i)} (\underline{t}_j - b_{ij}) - \underline{t}_i$$

where b_{ij} represents a valuation of the arc connecting i to j. Since the definition of free float is more restrictive than total float, the first is always a fraction of the second.

The *interfering float* is the difference between total float and free float. It characterizes the pairing of a task i scheduling with scheduling of following tasks in the graph:

$$ItFloat_i = TFloat_i - FFloat_i$$

The *independent float* of task i is the positive gap – if it exists – separating two dates:

– the earliest start time for i calculated in the case where all previous tasks on the graph are set as late as possible; and

– the latest start time for i calculated in the case where all following tasks on the graph are set as early as possible,

$$IdFloat_i = \min(0, \min_{j \in \Gamma^+(i)} (\underline{t}_j - b_{ij}) - \max_{l \in \Gamma^-(i)} (\bar{t}_l + b_{li}))$$

When the main requirement in a project is to end as soon as possible, it is advisable to schedule non-critical tasks earliest to offset possible contingencies during their execution as long as the available float is sufficient. Very often, however, considering other criteria linked to resources leads to delaying the start or prolonging the duration of non-critical tasks. We can use total float to classify tasks and reinforce operation control over those with little control because a too long delay can create a new critical path and prolong the project duration. We must keep in mind that any action on a task's total float can have consequences on the total

float of other tasks. The notion of free float proves to be more useful for the consideration of other criteria.

2.4. Shop scheduling

2.4.1. *Introduction*

In shop scheduling problems, resources are machines only able to execute one task – or operation – at a time. On the other hand, each job involves an indivisible physical entity, called *product*, or *lot* when several identical products are grouped. Since an entity cannot be in two places at the same time, a single job can only be executed one operation at a time on a single machine.

We will discuss one-machine and multi-machine problems consecutively. In the first case, each job is reduced to a single operation; we must then find a job sequence on the single machine. In the second case, we will first distinguish identical parallel machine problems which use the same single operation job hypothesis and for which the solution is a set of sequences, one for each machine. We will then study multi-machine shop problems where jobs include several operations, each requiring a specific machine to be available in a single copy. This case covers three types of problems, whether the sequence of operations for a single job is set and common to all jobs (*flow shop*), set but inherent to each job (*job shop*), or finally undetermined (*open shop*). Open shop problems are discussed in Chapter 10.

2.4.2. *Basic model*

2.4.2.1. *Data, variables and constraints*

The shop includes m machines. We must complete n jobs that can start at $t = 0$. Each job i is made up of n_i operations which must be executed in sequence. The j^{th} operation of job i is called *(i, j)*; it uses machine m_{ij}, with no interruption with a processing time p_{ij}, and the respect of production routes mandates $(i,1) \prec (i,2) \prec \ldots \prec (i,n_i)$ where $\prec$ symbolizes the relation of precedence. Setup times are independent of the sequence of operations and are included in their processing time.

The objective of the scheduling problem is to set start times t_{ij}. In order to do this, we must determine the order of circulation of all jobs for each machine, or *sequence*, since resources are disjunctive. From these sequences, several feasible

schedules – localizing in an absolute manner the operations in time – can be obtained according to the criterion to optimize.

2.4.2.2. *Criteria*

In shop scheduling, we attempt to minimize the maximum or the average over all jobs of indicators based on product wait times between two operations (flow time, delays, etc.). We often operate a weight on criteria (weight w_i) in order to model preferences or costs during execution of operations. Since several criteria are regular, the search for optimal solutions can focus on dominant left-shift schedule subsets.

2.4.3. *One-machine problem*

The study of single (each job only includes one operation to which it is assimilated) machine environments serves as a basis for developing more realistic, albeit more complex, multi-machine environment reasonings. Since the basic shop scheduling model is retained, we will first highlight the hypotheses inherent to a one-machine problem. The basic results are presented before addressing resolution methods which may also be used in the more general context of multi-machine problems.

2.4.3.1. *Specific results – priority rules*

For certain criteria, *list algorithms*, based on rules sequencing operations according to an imposed order of priority, make it possible to efficiently lead to an optimal resolution. The operations are sequenced from $t = 0$. At date t, and from all unscheduled operations that are *ready* ($\{i / \underline{t}_i \leq t\}$), we choose the one with the highest priority. The following moment of decision ($t' > t$) corresponds to the next greatest moment among the earliest end dates of scheduled operations (O) and the smallest of the earliest start times of operations not yet scheduled (N):

$$t' = \max\left(\min_{\substack{i \in O \\ \underline{t}_i + p_i > t}} (\underline{t}_i + p_i), \min_{j \in N} (\underline{t}_j) \right)$$

Priority may be linked to the shortest processing time, to the earliest due date, to the greatest quantity of work remaining, etc. (see Chapter 6). The advantage of priority rules resides in their simplicity of application. However, optimality is still dependent on the criterion chosen.

Sequencing of operations in increasing processing times/weights (p_i / w_i) ratio makes it possible to minimize the total flow time or work-in-process inventory. This

corresponds to respecting the WSPT rule (weighted shortest processing time first) or *Smith's rule* [SMI 56]. Sequencing operations in increasing processing times (SPT rule) minimizes the total lateness. Sequencing operations in increasing due dates minimizes maximum lateness and maximum tardiness. The corresponding rule is called EDD (earliest due date first) also known as *Jackson's rule* [JAC 55]. Note that the two previous rules can be adapted to optimally solve problems associated with the same criteria in the case where we consider different availability dates and where preemption is authorized [HOR 74, SMI 56]. The Hodgson-Moore algorithm minimizes the total number of tardy operations [MOR 68]. It consists of gradually building a schedule based on the EDD rule. When a delay appears in the partial schedule, the longest processing time operation already in place is pushed back to the end of the schedule. In the presence of precedence constraints between tasks, we have an exact polynomial method to minimize the maximum delay criteria. As in the application of Jackson's rule (EDD), the idea is to propagate the latest start time constraints with the Bellman algorithm and to place operations in increasing order of new due dates obtained.

For considering other criteria (for example the minimization of the total tardiness) under general conditions, the absence of specific results leads to the use of approximate methods (heuristic or metaheuristic, for example genetic algorithms – see Chapter 4) or general methods such as tree search procedures [CAR 82, MAC 93] (see also the work by Morton and Pentico [MOR 93]).

2.4.4. *Parallel machine problems*

With the one-machine problem, scheduling comes down to sequencing operations. The problem with parallel machines is characterized by the fact that several machines can be used for the execution of a job that only requires one of them. Theoretically, this problem is a generalization of the one-machine problem and a specific case of multi-machine shop problem. It is a problem frequently encountered in real applications, particularly in the context of *hybrid flow shops* (see Chapters 4 and 9), but also in the environment of computer processing scheduling [BLA 96a]. In this case, operations correspond to processing programs executed on processors. A task can start and then be interrupted if a more urgent process becomes available, which requires close attention to the question of preemption.

The resolution goes through the process of allocation decisions and is generally performed in two steps:

– decide on which machine each operation will be executed;

– determine the sequence of operations on each machine.

The problem is generally subdivided into three classes based on whether the machines are:

– identical: processing time is the same for all machines;

– uniform: processing time uniformly varies according to machine performance;

– independent: processing time is completely variable between the different machines.

We will only cover the case of identical machines considering the criteria of makespan and the sum of due dates.

2.4.4.1. *Makespan minimization*

Makespan minimization leads to a better load distribution for a cost effective use of plant capacity. In the absence of precedence constraints and in the case of interruptible operations, the MacNaughton algorithm provides a minimal time scheduling in linear time [MAC 59] (Algorithm 2.2).

If the preemption is prohibited, the problem becomes NP-hard with two machines or more; exact methods are mainly B&B procedures. We can use a heuristic which consists of placing operations according to LPT policy (longest processing time first) and assign them according to this order, consecutively to the machine with the lightest load. The shortest operations are then kept at the end of the schedule in order to balance the load as much as possible.

With precedence constraints, and in the specific case where the number of machines is at least equal to the number of jobs, we again fall back to the project scheduling problem and we can use the Bellman algorithm, otherwise some optimality results exist when the precedence graph has specific characteristics. The general problem is NP-hard in the strong sense ([PIN 95], page 66).

1. $C^*_{\max} = \max\left(\frac{1}{m}\sum_{i=1}^{n} p_i, \max_i p_i\right)$
2. *Select an operation and schedule it on the first machine at* $t = 0$.
3. *Schedule an operation not yet selected on the same machine as early as possible. Restart as long as total time on the machine does not exceed* $C^*_{\max}$.
4. *Transfer the part of the work scheduled after* $C^*_{\max}$ *onto the next machine. Go back to 3.*

Algorithm 2.2. *MacNaughton algorithm*

2.4.4.2. *Sum of due dates*

The minimization problem of the average due dates is optimally resolved with SPT. For the weighted average, WSPT-FAM (weighted shortest processing time on first available machine) is a good heuristic. In the presence of precedence constraints, the problem is NP-hard in the strong sense.

2.4.5. *Flow shop*

In the case of a flow shop, any job visits each machine in the shop and the routing of a job through the different machines is the same for all jobs (unidirectional flow). This single route is a part of the problem data. This specification is very often encountered in practice; it corresponds to an assembly or processing line for example. An important case in particular is the one of a simplified flow shop where the sequence – or permutation – of jobs visiting one machine is the same for all machines. This is the *permutation flow shop*. The *hybrid flow shop* case corresponds to a generalization of the flow shop and parallel machine problems. In this problem, the shop is made up of a certain number of ordered levels; each level has several parallel machines. This problem is discussed in Chapters 4 and 9. The specific results presented here exclusively involve the minimization objective of makespan and are limited to the permutation flow shop for which we obtain the most significant results. In addition, we know that schedules such that the job sequence is identical on the first two machines are dominant for regular criteria (see section 2.2.3.3).

2.4.5.1. *Two-machine flow-shop*

In this case, an optimal schedule for the makespan criterion can be searched for among permutation schedules. We apply the sufficient optimality condition given by *Johnson's rule* [JOH 54]: job i precedes job j in the optimal sequence $(i \prec j)$ if:

$$\min(p_{i1}, p_{j2}) \leq \min(p_{j1}, p_{i2}).$$

The associated algorithm [CAR 88] tends to put at the beginning of the schedule the jobs with lower processing times on the first machine, while rejecting the jobs with lower processing times on the second machine at the end of the schedule:

1. *Constitute job groups* $U = \{i \,/\, p_{i1} < p_{i2}\}; V = \{i \,/\, p_{i1} \geq p_{i2}\}$.
2. *Schedule* U *according to SPT sequence on the first machine (* p_{i1}*) and* V *according to the LPT sequence on the second machine (* p_{i2}*).*
3. *The optimal sequence is given by jobs scheduled in* U *followed by those scheduled in* V.

Algorithm 2.3. *Johnson algorithm*

2.4.5.2. *Three-machine flow-shop*

Schedules such that the job sequence is identical on the two last machines are dominant for the makespan criterion. Up to three machines, an optimal schedule can still be searched for among permutation schedules.

In the following two specific cases: $\min_i p_{i1} \geq \max_i p_{i2}$ or $\min_i p_{i3} \geq \max_i p_{i2}$ (the load on M_2 is always lower than on M_1 or M_3), we can extend Johnson's rule. They correspond to the fact that M_2 does not represent a "bottleneck", i.e. that it is completely dominated by M_1 or M_3. We then define two fictitious machines M'_1 and M'_3, on which respective processing times of a job are: $p'_{i1} = p_{i1} + p_{i2}$; $p'_{i2} = p_{i2} + p_{i3}$ and we apply Johnson's rule to this new two-machine problem.

Outside of these specific cases, the problem is NP-hard in the strong sense and we go back to the context of the *m*-machine problem. Works by [STO 63] and [IGN 65] respectively were among the first to propose a program in integer variables and a B&B for its resolution.

2.4.5.3. M-*machine flow-shop*

Generally, there is not an exact polynomial method. We use tree search methods, integer programming and heuristics, such as for example the famous Campbell, Dudek and Smith (CDS) method, a multi-step extension of Johnson's rule [CAM 70].

From the *m*-machine problem, we generate $m-1$ fictitious problems with two similar machines:

– at step 1, we only consider the first and last machine for a job i:

$$p_{i1}^{1} = p_{i1} \; ; \; p_{i2}^{1} = p_{im}$$

– at step 2, both equivalent machines are made up respectively of the first two and the last two machines regardless of any intermediate machine. Generally at step l, we have the following processing time on both equivalent machines:

$$p_{i1}^{l} = \sum_{k=1}^{l} p_{ik} \; ; \; p_{i2}^{l} = \sum_{k=m \; l+1}^{m} p_{ik} \; ; \; l = 1, \ldots, m-1$$

– for each of these fictitious problems, we determine the optimal sequence with the help of Johnson's rule;

– retain the best solution amongst the $m-1$ solutions obtained.

2.4.6. *Job shop*

The job shop problem can be found when a shop manufactures parts with different characteristics, but nevertheless requires processes where the sequences are known in advance. Each job must visit machines in a given sequence, but the difference with the flow shop is that this sequence may be different for each job (multidirectional flow).

In the following, we propose a quick overview of the most traditional results and methods. For a more detailed explanation of the job shop problem, refer to states of the art such as [BLA 96b] and [JAI 99]. As in the case with the flow shop, we will only consider makespan minimization here. There are very few specific results; they are limited to the case of the two-machine and two-job problems.

2.4.6.1. *Two-machine job-shop*

This problem has a known solution reachable in polynomial time. It is resolved by the Jackson algorithm, based on Johnson's rule [JAC 56, JOH 64]:

1. *Separate jobs into four sets:*
   – $\{O_1\}$ *: set of jobs made up of an operation executed on* M_1*;*
   – $\{O_2\}$ *: set of jobs made up of an operation executed on* M_2*;*
   – $\{O_{12}\}$ *: set of jobs made up of two operations, the first one on* M_1*, the second on* M_2*;*
   – $\{O_{21}\}$ *: set of jobs made up of two operations, the first one on* M_2*, the second on* M_1*.*
2. *Jobs for* $\{O_{12}\}$ *and* $\{O_{21}\}$ *are sequenced according to Johnson's rule; those for* $\{O_1\}$ *and* $\{O_2\}$ *do not have a specific sequence.*
3. *Optimal scheduling respects the following sequences:*
   – *for* M_1*:* $\{O_{12}\} \prec \{O_1\} \prec \{O_{21}\}$*;*
   – *for* M_2*:* $\{O_{21}\} \prec \{O_2\} \prec \{O_{12}\}$*.*

Algorithm 2.4. *Jackson algorithm*

2.4.6.2. *Two-job job-shop*

With the help of a graphical method, this problem can also be optimally solved in polynomial time [AKE 55, HAR 63]. We consider a plan made up of two time axes, with the abscissa associated with the execution of job 1 and the ordinate with the execution of job 2. The determination of a shortest time schedule goes back to the search for shortest routes in a plan paved with obstacles, insurmountable rectangles representing the simultaneous execution of both jobs on the same machine. The route includes horizontal lines (only job 1 executing), vertical lines (only job 2), or at 45° diagonal (jobs 1 and 2 simultaneously on two different machines) lines. The route maximizing the quantity of simultaneous jobs leads to the shortest makespan. Its value is given by the sum of the length of horizontal, vertical segments and projections of 45° segments on one of its axes.

2.4.6.3. M *machine job-shop*

In general, the problem is NP-hard in the strong sense. It is addressed by enumeration methods based on mixed-variable programming or by tree search methods, and heuristics based for example on priority rules. A more sophisticated modeling tool is called a *disjunctive graph* [ROY 70]. Operations are associated with nodes. The set of arcs is made up of a conjunctive part representing the precedence constraints linked to each single job (operating sequence) and a disjunctive part associated with all conflicts related to using resources that cannot be shared. A feasible schedule is characterized by the choice of one directed arc – or *arbitration* – in each disjunction pair. Feasible schedule makespan is the length of the longest path between the *beginning* node and *end* node on the graph where all

disjunction pairs have been arbitrated without generating a circuit. The detailed description of this model is given in section 10.6.

2.4.6.3.1. Generating schedules with priority rules

This approach consists of generating specific schedules (semi-active, active, non-delay) with the help of priority rules for the choice of candidate operations to be scheduled [BAK 74, CON 67]. An operation is "schedulable" if the previous operation in its production route is finished and if the machine it uses is idle. Its earliest start time is then the maximum date between the finish time of its immediate predecessor in the routing and the finish time of the last scheduled operation on the machine it uses. Two procedures are mainly used to constitute all schedulable operations:

– by generation of active schedules: among schedulable operations (initially the first for each job), we select the one for which the earliest finish time is the smallest; we call it c^*. All schedulable operations are made up of operations with an earliest start time lower than c^* on the machine where this operation may be executed;

– by generation of non-delay schedules: the procedure is similar to the previous one, except for the fact that the selected operation is the one for which the earliest start time is the smallest and called t^*. The generation of non-delay schedules ensures that a machine does not remain idle when an operation is waiting for this machine. However, there is no guarantee that all non-delay schedules contain an optimal solution to the scheduling problem raised, contrary to all active schedules.

Once the group of schedulable operations is completed, it is time to effectively select the operation to be executed on the machine concerned. In order to do this, we use priority rules such as SPT, LPT, EDD, FIFO, etc. (see Chapter 6). For a selected operation, we arbitrate machine usage conflicts by adding arcs between this operation and all operations still to be executed on this same machine on the disjunctive graph. The schedule generation approach can also be associated with a branch and bound procedure to optimally solve a job shop problem. At each step of this procedure, we can branch as soon as several operations become ready.

2.4.6.3.2. Shifting bottleneck heuristic

The shifting bottleneck heuristic is a powerful method for the resolution of job shop problems [ADA 88, DAU 93]. It is linked to the *shifting bottleneck* concept and uses the efficiency of a tree search procedure to solve the one-machine problem with minimization of the maximum lateness [CAR 82].

1. *M = group of machines in the problem*
2. *M_0 = group of machines on which conflicts have been solved (maybe temporarily); $M_0 = \varnothing$.*
3. *Choose in $M \setminus M_0$ the bottleneck machine. For this, we consider the associated conjunctive problem and for each machine in $M \setminus M_0$, we consider a one-machine problem with minimization of the maximum lateness that is solved by a B&B. The bottleneck machine is resource k for which we obtain the largest lateness.*
4. *Reusing the disjunctive graph and adding arcs corresponding to the optimal sequence on machine k.*
5. *Re-sequence all machines M_0 by including the sequence of operations associated with k and by resolving a new one-machine problem for each.*
6. *Reiterate for a new machine $M \setminus M_0$ (return to 3) up to $M_0 = M$.*

Algorithm 2.5. *Shifting bottleneck heuristic*

For examples on the application of this heuristic, we can refer to [PIN 99]. We also find a variation of this heuristic to solve the minimization of the total weighted tardiness.

2.5. Conclusion

This chapter has attempted to give a quick outline of methods which have now become standard for project and shop scheduling. For more detailed information on description and application of these methods, as well as others not described here, please refer to the numerous books on scheduling, among them: [BAK 74, BEL 82, BLA 07, CAR 88, CON 67, ESQ 99, FRE 82, LEU 04, PIN 95, PIN 05].

The objective here has mainly been to provide the necessary material for understanding the following chapters which describe specific methods and concepts in greater detail.

2.6. Bibliography

[ADA 88] ADAMS J., BALAS E. and ZAWACK D., "The shifting bottleneck procedure for job shop scheduling", *Management Science*, vol. 34, p. 391-401, 1988.

[AKE 56] AKERS S.B., "A graphical approach to production scheduling problems", *Operations Research*, vol. 4, p. 244-245, 1956.

[BAK 74] BAKER K.R., *Introduction to Sequencing and Scheduling*, John Wiley, New York, 1974.

[BEL 74] BELLMANN R. and DREYFUS S.E., *Applied Dynamic Programming*, Princeton University Press, 1974.

[BEL 82] BELLMANN R., ESOGBUE A.O. and NABESHIMA I., *Mathematical Aspects of Scheduling and Applications*, Pergamon Press, 1982.

[BER 87] BERTSEKAS D.P., *Dynamic Programming: Deterministic and Stochastic Models*, Prentice Hall, New Jersey, 1987.

[BLA 96a] BLAZEWICZ J., ECKER K.H., PESCH E., SCHMIDT G. and WEGLARZ J., *Scheduling Computer and Manufacturing Processes*, Springer, 1996.

[BLA 96b] BLAZEWICZ J., DOMSCHKE W. and PESCH E., "The job-shop scheduling problem: conventional and new solution techniques", *European Journal of Operational Research*, vol. 93, no. 1, p. 1-33, 1996.

[BLA 07] BLAZEWICZ J., ECKER K.H., PESCH E., SCHMIDT G. and WEGLARZ J., *Handbook on Scheduling*, Springer Verlag, Berlin, New York, 2007.

[CAM 70] CAMPBELL H. G., DUDEK R.A. and SMITH M.L., "A heuristic algorithm for the *n* job, *m* machine sequencing problem", *Management Science*, vol. 16, no. 10, p. 630-637, 1970.

[CAR 82] CARLIER J., "The one machine sequencing problem", *European Journal of Operational Research*, vol. 11, p. 42-47, 1982.

[CAR 88] CARLIER J. and CHRETIENNE P., *Problèmes d'ordonnancement: modélisation/ complexité/algorithmes*, Masson, Paris, 1988.

[CON 67] CONWAY R.W., MAXWELL W.L. and MILLER L.W., *Theory of Scheduling*, Addison Wesley, 1967.

[DAU 93] DAUZÈRE-PÉRÈS S. and LASSERRE J.-B., "A modified shifting bottleneck procedure for job shop scheduling", *International Journal of Production Research*, vol. 31, p. 923-932, 1993.

[ESQ 99] ESQUIROL P. and LOPEZ P., *L'ordonnancement*, Economica, Paris, 1999.

[FRE 82] FRENCH S., *Sequencing and Scheduling*, Ellis-Horwood, 1982.

[GAR 79] GAREY M.R. and JOHNSON D.S., *Computers and Intractability*, Freeman, 1979.

[HAR 63] HARDGRAVE W.W. and NEMHAUSER G.L., "A geometric model and graphical algorithm for a sequencing problem", *Operations Research*, vol. 12, no. 2, 1964.

[HOR 74] HORN W. A., "Some simple scheduling algorithms", *Naval Research Logistics Quarterly*, vol. 21, p. 177-185, 1974.

[IGN 65] IGNALL E.J. and SCHRAGE L.E., "Application of the branch and bound technique to some flow-shop scheduling problems", *Operations Research*, vol. 13, no. 13, 1965.

[JAC 55] JACKSON J.R., Scheduling a production line to minimize maximum tardiness, Management research project, University of California, Los Angeles, no. 43, 1955.

[JAC 56] JACKSON J.R., "An extension of Johnson's results on job-lot scheduling", *Naval Research Logistics Quarterly*, vol. 3, no. 3, 1956.

[JAI 99] JAIN A.S. and MEERAN S., "Deterministic job-shop scheduling: past, present and future", *European Journal of Operational Research*, vol. 113, p. 390-434, 1999.

[JOH 54] JOHNSON S.M., "Optimal two and three-stage production schedules with setup times included", *Naval Research Logistics Quarterly*, vol. 1, no. 1, p. 61-68, 1954.

[JOH 64] JOHNSON L.A. and MONTGOMERY D.C., "Operations scheduling", *Operations Research in Production Planning, Scheduling and Inventory Control*, John Wiley & Sons, 1964.

[LAW 66] LAWLER E.L. and WOOD D.E., "Branch-and-bound methods: A survey", *Operations Research*, vol. 14, p. 699-719, 1966.

[LEU 04] LEUNG J.Y-T., *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Chapman & Hall/CRC Computer & Information Science Series, 2004.

[MAC 59] MACNAUGHTON R., "Scheduling with deadlines and loss functions", *Management Science*, vol. 6, p. 1-12, 1959.

[MAC 93] MACCARTHY B. and LIU J., "Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling", *International Journal of Production Research*, vol. 31, p. 59-79, 1993.

[MOO 68] MOORE J.M., "An n job, one machine sequencing algorithm for minimizing the number of late jobs", *Management Science*, vol. 15, p. 102-109, 1968.

[MOR 93] MORTON T.E. and PENTICO D., *Heuristic Scheduling Systems*, John Wiley & Sons, 1993.

[NEM 88] NEMHAUSER G.L. and WOLSEY L., *Integer and Combinatorial Optimization*, John Wiley, New York, 1988.

[PIN 95] PINEDO M., *Scheduling: Theory, Algorithms and Systems*, Prentice Hall, 1995.

[PIN 99] PINEDO M. and CHAO X., *Operations Scheduling with Applications in Manufacturing and Services*, Irwin McGraw-Hill, 1999.

[PIN 05] PINEDO M., *Planning and Scheduling in Manufacturing and Services*, Springer, 2005.

[PRI 94] PRINS C., *Algorithmes de graphes*, Eyrolles, 1994.

[ROY 70] ROY B., *Algèbre moderne et théorie des graphes*, vol. II, Dunod, 1970.

[SMI 56] SMITH W.E., "Various optimizers for single-stage production", *Naval Research Logistics Quarterly*, vol. 3, p. 59-66, 1956.

[STO 63] STORY A.E. and WAGNER H.M., "Computational experience with integer programming for job-shop scheduling", Chapter 14 in *Industrial Scheduling*, Muth J.F. and Thompson G.L. (eds), Prentice Hall, 1963.

[WOL 98] WOLSEY L., *Integer Programming*, John Wiley, New York, 1998.

This page intentionally left blank

Chapter 3

Metaheuristics and Scheduling

3.1. Introduction

Scheduling involves taking decisions regarding the allocation of available capacity or resources (equipment, labor and space) to jobs, activities, tasks or customers over time. Scheduling thus results in a time-phased plan, or schedule of activities. The schedule indicates what is to be done, when, by whom and with what equipment. Although the statement of a manufacturing organizational problem such as scheduling may seem simple, we must never underestimate the effort necessary to find its solution [WID 98]. Scheduling problems can be modeled as assignment problems, which represent a large class of combinatorial optimization problems. In most of these cases, finding the optimal solution is very difficult. In fact, except for a few exceptions, the only known method to solve the problem to optimality would be to enumerate an exponential number of possible solutions! Specialists in this case speak of NP-complete problems [CAR 88, GAR 79]. In these conditions, it is necessary to find a solution method providing solutions of good quality in a reasonable amount of time: this is what heuristic methods are all about.

This chapter focuses on describing the three main heuristic classes, *constructive* methods, *local search* methods and *evolutionary* algorithms [COS 95c]. Since these methods are general enough to be applied to a multitude of combinatorial optimization problems, they are called *metaheuristics*.

This chapter is organized as follows: the next two sections indicate how to model scheduling problems in terms of combinatorial optimization and briefly describes

---

Chapter written by Marino WIDMER, Alain HERTZ and Daniel COSTA.

the major issues encountered when solving such problems. A complete presentation of the main metaheuristics is the subject of section 3.4, preceding a detailed analysis of the application of a tabu search method for the job shop scheduling problem with tooling constraints.

3.2. What is a combinatorial optimization problem?

Combinatorial optimization is the field of discrete mathematics involving the resolution of the following problem.

Let X be a set of solutions and f a function that measures the value of each solution in X. The objective is to determine a solution $s^* \in X$ minimizing f, i.e.:

$$f(s^*) = \min_{s \in X} f(s)$$

Set X is presumed finite and is in general defined by a set of constraints. As an example, for a job scheduling problem on one machine, X can be made up of all job sequences satisfying precedence and priority constraints while f can correspond to the date at which the last job is finished (makespan).

3.3. Solution methods for combinatorial optimization problems

Despite the permanent evolution of computers and the progress of information technology, there will always be a critical size for X above which even a partial listing of feasible solutions becomes prohibitive. Due to these issues, most combinatorial optimization specialists have focused their research on developing heuristic methods. A heuristic method is often defined [NIC 71] as a procedure that uses the structure of the considered problem in the best way possible in order to find a solution of reasonably good quality in as little computing time as possible. General properties of these techniques and the circumstances in which they apply are described in [MÜL 81, SIL 80].

Most difficult combinatorial optimization problems are NP-complete (see Chapter 2). Since no efficient algorithm for solving NP-complete problems is known despite the numerous efforts over the last 50 years, the use of heuristic methods is completely justified when facing such difficult problems. The performance of a heuristic method typically depends on the quality of the solution produced as output and on the computing time necessary to reach such a solution. A compromise has to be found on a case-by-case basis according to the considered optimization problem since these two criteria are of opposite nature. Since the optimal solution value of a

large scale difficult combinatorial optimization problem is typically unknown, the quality of a solution produced by an algorithm is typically evaluated using bounds or estimates of this optimal solution value. It might also be interesting to analyze the performance of a given heuristic method in the worst case scenario. A measure of the biggest possible error (with respect to the optimal value) is particularly useful when the goal is to specify in which circumstances the considered heuristic method should not be used.

Although obtaining an optimal solution is not guaranteed, the use of a heuristic method provides multiple advantages when compared with exact methods:

– the search for an optimal solution can be inappropriate in certain practical applications for many reasons such as the large size of the problem considered, the dynamic characterization of the work environment, a lack of precision in data collection, difficulties encountered when formulating the constraints in mathematical terms, the presence of contradictory objectives;

– when applicable, an exact method is often much slower than a heuristic method, which generates additional computing costs and a typically very long response time;

– research principles at the basis of a heuristic method are generally more accessible to inexperienced users. The lack of transparency that characterizes certain exact methods requires regular intervention from specialists or even from the method's designers;

– a heuristic method can easily be adapted or combined with other types of methods. This flexibility increases the range of problems to which heuristic methods can be applied.

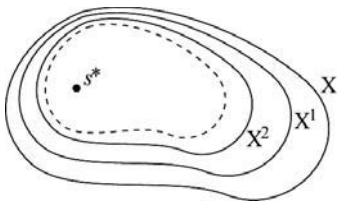
Even though a good knowledge of the problem to be solved is the main contributor to the efficient and successful use of a heuristic method, there are several general rules that can be used to guide the search in promising regions of the solution space X. Guidelines for the use of heuristics in combinatorial optimization can be found in [HER 03]. A classification of heuristic methods was proposed by Zanakis *et al.* [ZAN 89]. In the next sections of this chapter, we describe three fundamentally different heuristic approaches. The research principles of these approaches constitute a basis for several known heuristic methods such as the greedy algorithm, tabu search, simulated annealing and genetic algorithms. The Committee on the Next Decade of Operations Research [CON 88] declared in 1988 that these last three methods were very useful and promising for the solution of a large number of practical applications in the future. This prediction has turned out to be true. Although very general in their concept, these methods do require a large modeling effort if we wish to obtain good results.

3.4. The different metaheuristic types

3.4.1. *The constructive approach*

Constructive methods produce solutions in the form of $s = (x_1, x_2, ..., x_n)$, starting from an initial empty solution $s[0]$ and then inserting at each step k ($k = 1, ..., n$), a component $x_{o(k)}$ ($o(k) \in \{1, 2, ..., n\} \backslash \{o(1), o(2), ..., o(k-1)\}$) in the current partial solution $s[k-1]$. The decision that is taken at a given step is based on the index of the inserted component and on the value to give it. This decision is then never reassessed. The vector representation $s = (x_1, x_2, ..., x_n)$ is quite suitable in solutions for a general assignment problem. Vector positions correspond to objects, whereas each component x_i defines the resource allocated to object i.

The exploration of solution space X with a constructive method is represented in Figure 3.1. The goal is to decrease the size of the problem at each step, which means confining ourselves to an increasingly smaller subset $X^k \subseteq X$. A constructive method finds an optimal solution when each considered subset contains at least one optimal solution $s^* \in X$. Unfortunately, cases where such a condition is always fulfilled are rare. The majority of constructive methods are greedy: at each step, the current solution is completed in the best way possible without taking into consideration all consequences that this generates with regards to final solution cost. In this sense, greedy style methods are often seen as myopic.



$$X^k = \{s = (x_1, x_2, ..., x_n) \in X \mid \text{components } x_{o(1)}, x_{o(2)}, ..., x_{o(k)} \text{ are fixed}\} \quad (k = 1, \ldots, n)$$

Figure 3.1. *Exploration of X with a constructive approach*

Constructive methods are distinguished by their speed and great simplicity. Solutions are very quickly generated for a given problem without having to use highly sophisticated techniques. However, the main drawback with these methods resides unfortunately in the quality of the solutions obtained. The use of the best choice at each step is a strategy with potentially catastrophic long term effects. From a theoretical standpoint, obtaining an optimal solution is only ensured for problems accepting a formulation in terms of matroids [GON 85]. It is thus generally wise to implement procedures anticipating side effects and future consequences caused by decisions made during the completion of a partial solution.

We illustrate constructive methods with the algorithm by Nawaz *et al.* [NAW 83] summarized below and developed for the search of a minimal length sequence in a simple flow shop. The scheduling problem in a simple flow shop is a production problem in which n jobs must be executed following the same sequence on each of the m machines in the shop; these jobs all have the same process plan but not the same processing times. The processing time of job i on machine j is denoted by p_{ij} ($i = 1 \ldots n, j = 1 \ldots m$).

1. *for each job i (i = 1 ... n), set* $P_i \leftarrow \sum_{j=1}^{m} p_{ij}$
2. *sort the jobs into a list in descending order of* P_i
3. *take the first two jobs from the list at step 2 and find the best sequence for these two jobs by evaluating both scheduling possibilities. The relative positions of these two jobs cannot be modified during the next heuristic phases; set* $i \leftarrow 3$;
4. *take the job in* i^{th} *position from the list at step 2 and find the best sequence by inserting this job in one of the i possible positions among the jobs already placed;*
5. **if** $i < n$ **then** *set* $i \leftarrow i+1$ *and go to step 4;*
   otherwise *STOP: the sequence found is the NEH heuristic solution*

Algorithm 3.1. *NEH heuristic*

Nawaz *et al.*'s algorithm is based on the assumption that a job with a long total processing time has priority over a job with a shorter total processing time. The final sequence is found in a constructive way, by inserting a new job at each step and by finding the best possible position for this new job, without modifying the relative positions of the jobs inserted earlier (see Algorithm 3.1).

This heuristic provides a very good compromise between solution quality and computing time [WID 91b].

3.4.2. *Local search approach*

Local search methods are iterative algorithms which explore the solution space X by moving step by step from one solution to another. This type of method begins with a solution $s_0 \in X$ arbitrarily chosen or otherwise obtained from a constructive method. The transition from one solution to another is made according to a series of basic modifications which must be defined on a case-by-case basis. The following notations are taken from the [WER 89] reference. We denote by Δ the set of all possible modifications that can be applied to a solution. A solution s' obtained from s by applying a modification $\delta \in \Delta$ is denoted as $s' = s \oplus \delta$. The *neighborhood* $N(s)$

of a solution $s \in X$ is defined as the set of solutions that can be obtained from s by applying a modification $\delta \in \Delta$. Specifically, we can write: $N(s) = \{s' \in X \mid \exists\, \delta \in \Delta: s' = s \oplus \delta\}$. This type of examination process is interrupted when one or more stopping criteria are met. Figure 3.2 illustrated the general scheme of a local search method. Consecutive transitions from one solution to a neighbor solution define a path in the solution space X.

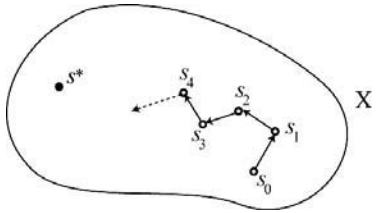


Figure 3.2. *Exploration of X with a local search approach*

An oriented graph **G**, called *state space graph*, can be defined to represent a local search. Each solution in X is associated with a vertex in **G**, and an arc from vertex s_1 to vertex s_2 is introduced in **G** if and only if s_2 is a neighbor of s_1 (i.e. $s_2 \in N(s_1)$). Modeling an optimization problem and choosing a neighborhood $N(s)$ must be done in such a way that for each solution $s \in X$, there is at least one path in **G** linking s to an optimal solution s^*. The existence of such paths ensures that the local search method can possibly reach an optimal solution starting from any initial solution s_0.

The neighborhood of a solution s in the context of assignment problems can be defined in a very simple way [FER 96]. Given a set of objects and a set of resources, and assuming that exactly one resource must be assigned to each object, we can define the neighborhood $N(s)$ of s as the set of solutions that can be obtained from s by changing the resource assignment of exactly one object.

The descent method described in Algorithm 3.2 is a first example of a local search method. This type of method moves in X by choosing at each step the best neighbor solution in the neighborhood $N(s)$ of the current solution s. This process is repeated as long as the value of the objective function decreases. The search stops when a local minimum is reached. Historically, descent methods have always been among the most popular heuristic methods to handle combinatorial optimization problems. However, they contain two major obstacles which considerably limit their efficiency:

– according to the size and structure of the considered neighborhood N(s), the search for the best neighbor solution in N(s) is a problem which can be as difficult as the original problem (which is to find the best solution in X);

– a descent method is unable to escape the first local minimum encountered. Combinatorial optimization problems however typically contain numerous local *optima* with an objective function value which can be very far from the optimal value. This is illustrated in Figure 3.3.

In order to cope with these deficiencies, more sophisticated local search methods have been developed in the last 20 years. These methods accept neighbor solutions with a lesser quality than the current solution in order to avoid local minima of function f. The most famous methods of this type are presented in the next sections. The main differences between these methods concern the rule used to choose a neighbor solution and the stopping criteria. The search can be stopped when a solution deemed close enough to being optimal is found. Unfortunately, the optimal value of a combinatorial optimization problem is typically not known, and it is therefore often not possible to use such a stopping criterion.

Initialization
  choose an initial solution $s \in X$;
  set $s^* \leftarrow s$;
Iterative process
  repeat
    generate N(s);
    determine $s' \in N(s)$ *such that* $f(s') = \min_{s'' \in N(s)} f(s'')$;
    set $s \leftarrow s'$;
    if $f(s) < f(s^*)$ **then** *set* $s^* \leftarrow s$;
  until $s \neq s^*$

Algorithm 3.2. *The descent method*

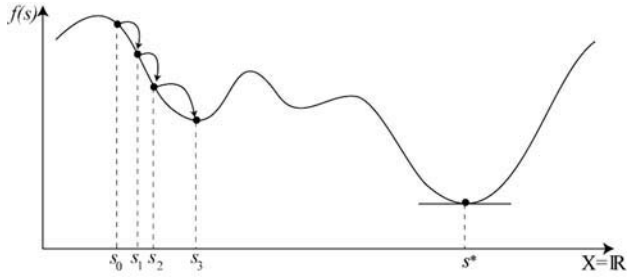


Figure 3.3. *A descent method cannot escape a local minimum*

The methods presented below are generally much more powerful than a simple descent method but also much more expensive in terms of computing resources. Their implementation must consider the maximum response time authorized by the program user. We should note in conclusion that significant effort is necessary to correctly adjust the parameters used by these methods in order to effectively guide the search throughout X.

3.4.2.1. *Simulated annealing*

Originally, the simulated annealing method goes back to experiments by Metropolis *et al.* [MET 53]. Their studies have led to a simple algorithm to simulate the evolution of an unstable physical system toward a state of thermal balance with a fixed temperature *t*. The state of the physical system is characterized by the exact position of all the atoms it contains. Metropolis *et al.* use a Monte Carlo method to generate a series of consecutive system states starting with a given initial state. Any new state is obtained when an atom executes an infinitesimal random movement. Let ΔE be the energy difference generated by such a disruption. The new state is accepted if the system energy decreases ($\Delta E < 0$). In the contrary case of ($\Delta E \geq 0$), it is accepted with a certain probability:

$$\text{prob}(\Delta E, t) = \exp\left(\frac{-\Delta E}{k_B \cdot t}\right)$$

where t is the system temperature and k_B a physical constant known as the Boltzmann constant. At each step, acceptance of a new state where the energy is not lower than the current state is decided by randomly generating a number $q \in [0,1[$. If q is lower than or equal to prob(ΔE,t), then the new state is accepted, otherwise the current state is maintained. Metropolis *et al*. have shown that a repeated use of this type of rule brings the system to a state of thermal balance. Many years passed after these studies from Metropolis *et al.* before the simulation algorithm they developed was used to define a new heuristic method for the resolution of a combinatorial optimization problem. Simulated annealing is a local search method where the search mechanism is modeled on the Metropolis *et al*. algorithm and principles of thermodynamic annealing. The idea consists of using the Metropolis *et al.* algorithm with decreasing temperature values *t*. The progressive cooling of a particle system is simulated by, on the one hand, making an analogy between the system energy and the objective function of a combinatorial optimization problem, and between system states and the solutions of the considered problem on the other hand. To reach states with as little energy as possible, the system is initially brought to a very high temperature and then slowly cooled down. When the temperature goes down, atom movements become less random and the system will tend to be in low energy states. System cooling must be carried out very slowly in order to reach a state of balance

at each temperature t. When no new state is accepted at a given temperature t, the system is considered frozen and it is presumed that it has reached a minimum level of energy.

Kirkpatrick *et al*. [KIR 83] and Cerny [CER 85] were the first to follow such a technique to solve combinatorial optimization problems. The neighborhood N(s) of a solution $s \in X$ contains all states which can be obtained from the current state by slightly moving one atom of the physical system. Only one neighbor solution $s' \in$ N(s) is generated at each iteration. This solution is accepted if it is better than the current solution s. Otherwise, we proceed as with the Metropolis *et al.* algorithm and the new s' solution is accepted with a probability prob($\Delta f,t$) depending on the importance of the damage $\Delta f = f(s') - f(s)$ and on a parameter t corresponding to the temperature. Changes in temperature are performed on the basis of a precise cooling pattern. Generally speaking, the temperature is lowered in steps each time a specific number of iterations are executed. The best solution found is stored in variable s^*. The algorithm is interrupted when no neighbor solution has been accepted during a complete constant temperature iteration cycle. The simulated annealing method is described in Algorithm 3.3.

Simulated annealing performance is closely linked to the cooling pattern involved. Numerous theoretical studies have been carried out on this subject and several variations have been proposed [COL 88, OSM 94]. Simulated annealing can be described in the form of a non-homogenous Markov chain [AAR 89, VAN 87], which leads to interesting results in terms of asymptotic convergence of the algorithm when certain specific conditions are met. Geman *et al.* [GEM 84] have demonstrated that the method converges in terms of probability to an optimal solution if temperature t_k at the k^{th} iteration meets the two following conditions:

i) $\lim_{k \to \infty} t_k = 0$

ii) $t_k \geq \dfrac{c}{\log(k+1)} \quad \forall\, k = 1, 2, \ldots$

where c is a constant independent of k. Hajek [HAJ 88] established a similar convergence result by asserting:

iii) $t_k = \dfrac{c}{\log(k+1)}$ and $c = \max_{s \in X} f(s) - \min_{s \in O} f(s)$

where O represents the set of all local minima of f which are not optimal solutions. In general, constant c is called the maximum depth of function f.

In reality, the above condition is difficult to obtain and very costly in terms of calculation. We generally prefer the cooling pattern by levels shown in Algorithm 3.3, even though it does not guarantee algorithm convergence to an optimal solution. Recently, the simulated annealing algorithm has been frequently used to solve practical optimization problems. A detailed review of the literature was carried out by Collins *et al.* [COL 88]. Several simulated annealing algorithm applications are proposed in a book edited by Vidal [VID 93]. The interested reader can refer to [EGL 90, REE 93, RUT 89, VAN 87] for more detailed information on simulated annealing.

```
Initialization
    choose an initial solution s ∈ X;
    set s*← s;
    set k← 0;                       (global iteration counter)
    set new_cycle← true;            (Boolean variable indicating if it is worth
                                    executing a new cycle of iterations)
    set t← t0;                      (t0 = initial system temperature)
Iterative process
    while new_cycle = true do
    set nbiter← 0;                  (iteration counter internal to a cycle)
    set new_cycle← false;
    while (nbiter < nbiter_cycle) do
        set k← k + 1 and nbiter← nbiter + 1;
        randomly generate a solution s'∈N(s);
        set Δf← f(s') – f(s);
        if Δf < 0 then set s← s' and new_cycle← true;
        otherwise
            set prob(Δf,t) ← exp(-Δf/t);
            generate a random real number q from an uniform distribution over the
            interval [0,1[;
            if q < prob(Δf, t) then set s← s' and new_cycle← true;
        if f(s) < f(s*) then set s*← s;
    set t: = a · t                  (0 < a < 1: cooling factor)
```

Algorithm 3.3. *Simulated annealing*

3.4.2.2. *Threshold accepting methods*

Threshold accepting methods are local search methods that can be seen as deterministic variants of simulated annealing. The main difference between these two methods is the level of acceptance of a lower quality solution at each step. In a threshold accepting method, this type of decision is taken in a deterministic manner, without having to use the principles of thermodynamic annealing. It is strictly based

on an auxiliary function $\gamma(s,s')$ and on a threshold S which can possibly involve the value $f(s^*)$ of the best solution encountered so far. Function $\gamma(s,s')$ and threshold S can be defined in many ways leading to several variations for threshold accepting methods. The threshold accepting method is presented in a generic way in Algorithm 3.4, followed by three adaptations of this method developed by Dueck and Scheuer [DUE 90, DUE 93]. The algorithm is generally interrupted when a number *nbiter_tot* of iterations is reached or when the best found solution s^* is not improved during a number *nbiter_max* of iterations.

Standard threshold accepting method

Threshold S is defined the same way as temperature t in the simulated annealing algorithm. It is initially set at a high value, then proportionally lowered (i.e., S is set equal to $a \cdot S$ with $0 < a < 1$) each time a predetermined number of iterations is completed. The aim of this method is to accept all neighbors s' which do not deteriorate the quality of the current solution s in a significant way. For this purpose, the auxiliary function $\gamma(s,s')$ is defined as $\gamma(s,s') = f(s') - f(s)$.

Great deluge method

In this case, threshold S corresponds to a level of water. In a minimization problem, a neighbor solution s' is accepted if and only if $f(s')$ is below the water level S regardless of the value $f(s)$ of the current solution. In this way, $\gamma(s,s') = f(s')$, The water level is initially set at a high value S_0, and it then decreases linearly (i.e., S is set equal to $S - d$ where d is a parameter measuring the decreasing speed).

```
Initialization
    choose an initial solution s ∈ X;
    set s*← s;
    set nbiter← 0;          (iteration counter)
    set best_iter← 0;       (iteration leading to the best s* solution found to date)
    set S← S0;              (S0 = initial threshold)
Iterative process
    while no stopping criterion is met do
        set nbiter← nbiter + 1;
        randomly generate a solution s' ∈ N(s);
        if γ(s,s') < S then
            set s← s';
            if f(s) < f(s*) then set s*← s and best_iter← nbiter;
        update threshold S
```

Algorithm 3.4. *Threshold accepting method*

Originally, this method was designed to solve maximization problems. First, the level of water is arbitrarily set at a low value, and it is then linearly increased. In such a case, a neighbor solution *s'* is only retained if it is over the water level (i.e., if $f(s') > S$). This explains the name of the method. Such a research principle can be simply illustrated by considering a hiker unable to swim and wanting to reach the highest point of a given region when the level of water constantly increases. The hiker's chances of success are higher if the water increase is slow (i.e., *d* is small).

Record-to-record travel method

In this adaptation of the threshold accepting method, all neighbor solutions *s'* of significantly lower quality than the best solution *s** found so far are rejected. In order to do this, we must define a maximal deterioration *max_d*, in relation to value $f(s^*)$, over which any neighbor solution *s'* is automatically rejected. As in the previous method, acceptance of a new solution is made without examining the value of the current solution, hence $\gamma(s,s') = f(s')$. Threshold S is initialized at a high value and is then updated at each step by setting S equal to $f(s^*) + max\_d$. Parameter *max_d* is typically not modified during the execution of the algorithm.

According to [DUE 90, DUE 93], the three previous methods provide numerous advantages compared to the simulated annealing method. Several tests were carried out in the context of the traveling salesman problem. It would seem that a threshold accepting method generally gives better results than simulated annealing while being quicker and less sensitive to the set of parameters used. Sinclair [SIN 93] confirmed these observations after using the great deluge and the record-to-record travel methods for balancing hydraulic turbine runners. We do not know of any more comparative studies leading to similar conclusions.

3.4.2.3. *Tabu search*

Tabu search is a general iterative method for combinatorial optimization introduced by Glover [GLO 86] in a specific context and later developed in a more general context [GLO 89, GLO 90]. Independently, Hansen developed the SAMD method (for Steepest Ascent Mildest Descent) based on similar ideas [HAN 86]. The tabu search method is very powerful over a considerable number of combinatorial optimization problems, particularly scheduling.

Method description

As previously mentioned, the movement from a current solution *s* to a neighbor solution *s'* is chosen in such a way that:

$$f(s') = \min_{s'' \in N(s)} f(s'')$$

As long as we are not in a local *optimum*, any iterative local search method behaves as the descent method and improves the objective function value at each step. On the other hand, when we reach a local *optimum*, the movement rule described earlier makes it possible to choose the lesser of bad neighbors, i.e. the one inducing as little increase of the objective function as possible. The disadvantage that would represent a method only based on this principle is that if a local minimum s is found at the bottom of a deep valley, it would be impossible to get out of it in a single iteration, and a movement from s to a neighbor solution $s' \in N(s)$ with $f(s') > f(s)$ can cause the opposite movement at the next iteration, since generally $s \in N(s')$ and $f(s) < f(s')$; there is therefore a risk of "cycling" around this local minimum s. For this reason, the tabu search algorithm uses a second principle consisting of keeping in the memory the last visited solutions, and prohibiting a return to them for a fixed number of iterations, the goal being to offer enough time to the algorithm to exit a local minimum. In other words, the tabu search method keeps a list T of "tabu" solutions at each step, and it is prohibited to move to any solution in T. When moving from s to a neighbor solution, the oldest element in T is removed and replaced by s. The necessary space to register a list of tabu solutions can be quite large in terms of memory. For this reason, it is sometimes preferable to prohibit a set of movements that would bring us back to solutions already visited. These prohibited movements are called *tabu movements*.

The aspiration function

During the choice for the best solution $s' \in N(s)$, we may have to decide between several candidates providing the same value to the objective function. If the chosen neighbor does not lead to a good region of the solution space X, it might then be desirable to be able to go back to a visited solution s despite the fact that it is now part of the tabu list T, in order to explore a new region neighboring s.

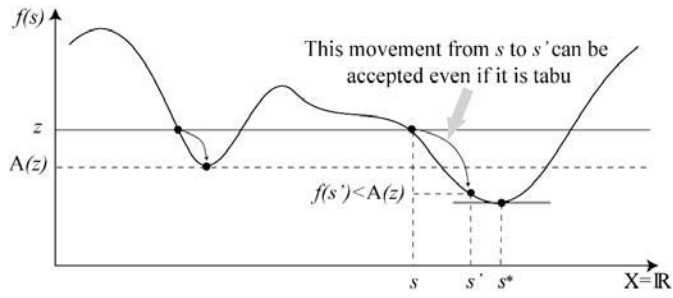


Figure 3.4. *Illustration of the aspiration function concept*

For this reason, the tabu search method involves a new ingredient called *aspiration function* defined over all values of the objective function: when a neighbor solution $s' \in N(s)$ belongs to T and meets the aspiration criterion (i.e. $f(s') < A(f(s))$), we cancel its tabu status and s' then becomes a candidate during the selection for the best neighbor of s. In general, $A(f(s))$ takes on the value of the best solution s^* encountered so far (i.e., we "aspire" to determine a better solution than s^*). Another possibility is to set $A(z)$ equal to the best value obtained by moving from a solution s with value $f(s)=z$ (see Figure 3.4).

Neighborhood

For certain problems, the neighborhood $N(s)$ of a solution s is large and, in addition, the only way to determine a solution s' minimizing f over $N(s)$ is to review all solutions in $N(s)$; we thus prefer to generate a subset $N' \subseteq N(s)$ only containing a sample of solutions neighboring s and we choose a solution $s' \in N'$ of minimal value $f(s')$.

Ending condition

We must still define a stopping condition. We are generally given a maximum number *nbmax* of iterations between two improvements of the best solution s^* encountered so far. In certain cases, it is possible to determine a lower bound $\underline{f}$ on the optimal value and we can then stop the search when we have found a solution s of value $f(s)$ close to $\underline{f}$.

The tabu search method is summarized in Algorithm 3.5.

Initialization
  Choose an initial solution $s \in X$;
  set $s^* \leftarrow s$;
  set $nbiter \leftarrow 0$; (iteration counters)
  set $T \leftarrow \varnothing$; (the tabu list is initially empty)
  initialize the aspiration function A;
  set $best\_iter \leftarrow 0$; (iteration where s^* was found)
Iterative process
  while $(f(s) > \underline{f})$ ***and*** $(nbiter - best\_iter < nbmax)$ **do**
  set $nbiter \leftarrow nbiter+1$;
  generate a subset $N' \subseteq N(s)$ *of solutions neighboring s;*
  choose the best solution $s' \in N'$ *solution such that* $f(s') \leq A(f(s))$ *or* $s' \notin T$;
  update the aspiration function A and the tabu list T;
  set $s \leftarrow s'$;
  if $f(s) < f(s^*)$ **then** *set* $s^* \leftarrow s$ *and* $best\_iter \leftarrow nbiter$

Algorithm 3.5. *Tabu search*

In summary, the following elements are the main ingredients of a tabu search method:

X	the set of solutions
f	the objective function defined on X
N(s)	the neighborhood of a solution $s \in X$
|T|	the number of tabu solutions or the size of the *tabu list* T
A	the aspiration function
nbmax	the maximum number of iterations between two improvements of s^*
N'	the subset of N(s) (the way of generating it and its size)
$\underline{f}$	a lower bound on the objective value.

Most tabu method ingredients are illustrated in section 3.5. Those not present have been considered as useless during the adaptation of the tabu method to the considered scheduling problems. Any reader interested in more illustrations of all ingredients can refer to scientific papers from Glover [GLO 89; GLO 90; GLO 97]. More refined versions of the tabu search algorithm have been presented in more detail in the literature. The interested reader should consult [GLO 89, GLO 90, GLO 93b, HER 97, REE 93, SOR 94, TAI 93, WER 89] for more information on this subject. Several strategies have been proposed to improve the efficiency of the tabu search algorithm presented above [GLO 97]. *Intensification* and *diversification* are two of these strategies.

Intensification consists of a detailed exploration of a region of X deemed promising. Its implementation usually resides in a temporary widening of the neighborhood of the current solution in order to visit a set of solutions sharing common properties. Another possibility consists of returning to the best solution s* encountered so far and to restart the search from this solution with a smaller tabu list size for a limited number of iterations. Diversification is a complementary technique to intensification. Its objective is to direct the search procedure to unexplored regions in X. The simplest diversification strategy is to restart the search process periodically from a solution either randomly generated or judiciously chosen in a region that has not yet been visited. In this way, we decrease the influence of the choice of the initial solution s_0 over the global algorithm performance. The same type of effect can be obtained by temporarily modifying the objective function or by favoring modifications not made during a large number of iterations.

To conclude this presentation of the tabu search method, we should mention that no theoretical result guarantees the convergence of the algorithm toward an optimal solution of the considered problem contrary to the simulated annealing method. The main reason for this fact comes from the nature of the method itself. Since the method is highly adaptive and flexible, its analysis with traditional mathematical

tools is difficult. The only known theoretical results to this day involve a probabilistic version of the tabu search algorithm close to the simulated annealing method. Faigle and Kern [FAI 92] have shown that it is possible to modify the objective function and the generation probability of neighbor solutions in order to ensure global convergence of the search process. Such a result is unfortunately not very useful in practice because the probability of convergence to an optimal solution only happens after an infinite time period. In addition, the tabu search method does not need to converge toward an optimal solution to be effective. The goal is to visit at least one optimal solution or a good quality solution during the search.

Despite the fact that no substantial theoretical result was established, the tabu search algorithm has been arousing increased interest since its discovery 20 years ago. Impressive results have been obtained for numerous combinatorial optimization problems. Several examples of tabu search applications are proposed in [GLO 93a]. A detailed example of a tabu search application is the subject of section 3.5, job shop scheduling with tooling constraints. For the conclusion of this section, we should note that Glass and Potts have achieved an interesting comparison of different local search methods for the single flow shop problem [GLA 96], while the same type of analysis was performed by Galinier and Hertz for the graph coloring problem [GAL 06].

3.4.3. *The evolutionary approach*

Life sciences and natural processes have always fascinated engineers. They use structures and mechanisms from reality to develop artificial objects used in various contexts. In the field of combinatorial optimization, natural phenomena complexity has served as a model for increasingly sophisticated algorithms in the last 35 years. The evolutionary methods presented in this section constitute the basis of a constantly growing field of computer programming.

Contrary to constructive and local search methods involving a single solution (partial or not), evolutionary methods handle a group of solutions at each search process step. The main idea is to regularly use the collective properties of a group of solutions, called population solutions, with the goal of efficiently guiding the search to appropriate solutions in the solution space X. In general, the size p of a population remains constant throughout the process. After the random or constructive method generation of an initial population containing solutions $s_{0,i} \in X$ ($i = 1, ..., p$), an evolutionary method attempts to improve the average quality of the current population by using natural evolution principles. In our terminology, the cyclic process at the basis of an evolutionary method is made up of the continuous succession of a cooperation phase and an individual adaptation phase. This new formalism applies to most evolutionary methods developed to date.

In the cooperation phase, the solutions in the current population are compared and then combined together in order to produce new and good quality solutions for the long term. The resulting information exchange leads to the creation of new solutions which have the predominant characteristics contained in current population solutions. In the individual adaptation phase, the solutions evolve independently respecting a series of predefined rules. Modifications experienced by each solution are performed without interaction with other solutions in the population. A new population of solutions is created at the end of each individual adaptation phase.

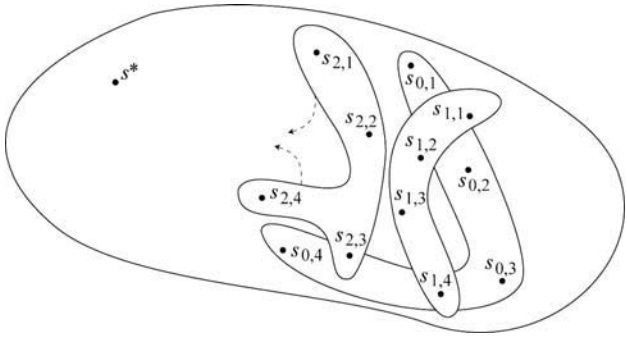


Figure 3.5. *Exploration of X with an evolutionary approach*

The search mechanism at the basis of an evolutionary method is briefly represented in Figure 3.5. The goal is to locate the best possible solutions by manipulating at each step a set of solutions located in different promising regions of the solution space X.

In what follows, we say that an evolutionary method prematurely converges or goes through a diversity crisis when the current population contains a high percentage of identical solutions. Different mechanisms can be incorporated to avoid this drawback. The simplest way to prevent premature convergence risks is to introduce a measure $E \in [0, 1]$ based on the notion of entropy [FLE 94, GRE 87] in order to evaluate the degree of diversity within a population of solutions. A value $E = 0$ indicates that the population is made up of identical individuals whereas a value closer to 1 suggests a large diversity in the population. When entropy E is judged to be too low at a given step, the idea is to take measures to reintroduce sufficient diversity in the current population.

3.4.3.1. *Genetic algorithms*

Genetic algorithms are evolutionary methods greatly influenced by biological mechanisms linked to the principles of natural evolution and selection. Initially

developed by Holland *et al.* [HOL 75] to respond to specific needs in biology, genetic algorithms have quickly been adapted to a large variety of contexts. In a simple genetic algorithm [GOL 87], the search is handled by three operators applied consecutively. The cooperation phase is managed by a reproduction operator and a crossover operator whereas the individual adaptation phase uses a mutation operator. It is important to note that concepts at the basis of genetic algorithms are extremely simple. In fact, they only involve randomly generated numbers and a set of generic probabilistic rules which do not necessarily consider all characteristics of the problem discussed. Chapter 4 focuses on genetic algorithms.

3.4.3.2. *Scatter search method*

The scatter search method [GLO 94, GLO 95] simulates an evolution which is not directly related to genetics. No restriction is made on how to code solutions. The cooperation phase does not refer to a reproduction operator, and the combination of solutions is more generic than with a genetic algorithm. The combination operator generates new individuals by considering groups of solutions with possibly more than two elements. The role of the individual adaptation phase is to repair individuals produced by the combination operator if they do not satisfy all constraints of the considered problem; during the individual adaptation phase, each individual is also possibly improved by means of an improvement operator. This type of mechanism can be implemented in the form of a local search method involving an auxiliary function penalizing individuals that do not satisfy all constraints of the considered problem. The few variations of scatter search proposed in literature were all perceived to be a generalization of a genetic algorithm [RAD 94]. Scatter search was originally developed by Glover [GLO 77] for the solution of integer programming problems. The idea is to continuously operate linear combinations over a set of solutions coded as vectors with integer components. An auxiliary procedure is applied at each step to repair individuals if the linear combination has produced non-integer components in vectors. This repairing process can be done by using systematic rounding or, even better, a local search method meant to find an integer solution that is the “closest” to the considered non-integer vector. Among successful applications of the scatter search method, we should mention the adaptations made for quadratic assignment problems [CUN 97], neural network training [KEL 96] or unconstrained continuous optimization [FLE 96a].

3.4.3.3. *The ant algorithm*

Collective performance [THE 94] of social insects such as ants, bees, wasps or termites has intrigued entomologists for a very long time. The main question involves mechanisms enabling individuals of a single colony to manage their activities and to favor survival of the species. Everything is done as if an invisible agent is coordinating activities of all individuals from the center of the colony.

Studies have shown that this global behavior was the result of a multitude of particularly simple local interactions. The nature of these interactions, information processing mechanisms and the difference between solitary and social behaviors has long remained a mystery. During the accomplishment of a specific task by an insect colony, it was observed that task coordination did not depend directly on workers but on the state of the task's progress. The worker does not manage its job; it is guided by it. Any working insect modifies the form of stimulation generating its behavior and causes the creation of a new stimulation that will trigger other reactions from it or a coworker.

To illustrate the appearance of collective structures in a society of insects, we must mention the example of an ant colony searching for a food source. Initially, ants leave their nest and move randomly. When an ant happens to discover a food source, it informs its coworkers of its discovery by setting off a temporary mark on the ground upon its return to the nest. This mark is nothing more than a chemical substance called a pheromone, which will guide the other ants toward the same food source. Upon their return, these ants also set off pheromones on the ground and thus reinforce the trail marking leading from the nest to the source of discovered food. Pheromone marking reinforcement in the most used trail optimizes food gathering. In the long run, ants will only use the closest source because the trail leading to remote sources will evaporate and become undetectable. This example shows that the ant colony converges to an optimal solution where each ant only has access to local information and it is unable to resolve the problem on its own in a reasonable amount of time.

In the last few years, engineers have focused on social insect behavior in order to create a new form of "collective problem solution". The ant algorithm, initially developed by Colorni *et al.* [COL 91, COL 92, DOR 96], is an evolutionary method where search mechanisms are greatly influenced by the collective behavior of an ant colony. In the cooperation phase, each current population solution is examined in turn in order to update a global memory. Then, the individual adaptation phase involves a constructive method which uses information contained in the global memory to create new solutions. This type of approach repeatedly uses a constructive method by including the accumulated experience from previous method applications. Each constructive method application corresponds to the work accomplished by a single ant. In the previous example, the global memory appears in the traces of pheromone set off by the ants. This search principle easily adapts to the general assignment problem. Concepts presented below generalize the ideas of Colorni *et al*. [COL 91, COL 92, DOR 96] in the traveling salesman context. Consider a set of n objects to which a resource $j \in \{1, ..., m\}$ has to be assigned. This assignment must meet a series of given constraints and be as inexpensive as possible. Two decisions are made at each step of a constructive method. The first one involves the choice of one of the n objects and the second defines the resource

assigned to it. In a traditional constructive method, these decisions are made in a manner that will complete the current solution in the best way possible. In other words, they are based on an object and a resource with the maximum appeal for the current step. The ant algorithm uses a complementary notion to an object or resource appeal. Partial solutions are completed in a probabilistic manner by relying on past experience. The term trace will then be used to represent an element of information which is known based on experiments carried out on the completion of previous solutions. At each step k of the development process, an ant chooses an object i, not yet assigned, with probability $p_{\text{obj}}(k, i)$ and a resource j that is feasible for object i, with probability $p_{\text{res}}(k, i, j)$. Each of these probabilities involves two numbers $\tau(s[k-1],.)$ and $\eta(s[k-1],.)$ which measure the trace and appeal respectively for an object or resource, given a partial solution $s[k-1]$ in which objects $o(1)$, ..., $o(k-1)$ have already been assigned to resources $x_{o(1)}$, ..., $x_{o(k-1)}$:

$$p_{\text{obj}}(k,i) = \frac{\tau_1(s[k-1],i) \,.\, \eta_1(s[k-1],i)}{\sum_{q \notin \{o(1),o(2),\ldots,o(k-1)\}} \tau_1(s[k-1],q) \,.\, \eta_1(s[k-1],q)}$$

$$p_{\text{res}}(k,i,j) = \frac{\tau_2(s[k-1],i,j) \,.\, \eta_2(s[k-1],i,j)}{\sum_{r \text{ is admissible for } i} \tau_2(s[k-1],i,r) \,.\, \eta_2(s[k-1],i,r)}$$

Algorithm 3.6 presents an ant algorithm for the solution of general assignment problems. The number of ants is n. During the first cycle all traces are set to 1 because the ants do not know the problem to be solved. At the end of a cycle, traces τ_1 and τ_2 are updates in relation to the characteristics of a solution produced by each ant. Generally, only one of the two decisions is made in a probabilistic way at each step of the constructive method. In most applications solved with the help of the ant algorithm, choosing the next object is done either naturally, or based on a deterministic rule not involving the concept of trace.

A simplified version of the ant algorithm was proposed by Colorni *et al.* [COL 94] for the job shop scheduling problem. They construct a graph G_{jobshop} based on the method by Roy and Sussmann [ROY 64]: each operation to be performed on the machines is a vertex of G_{jobshop}; an artificial vertex (corresponding to the start of the schedule) is added to G_{jobshop}, and linked by an arc to each job's first operation; for each job, two consecutive operations in the process plan are linked by an arc; two operations, which can be executed on a single machine, are connected by an *edge* (i.e., a non-oriented arc), then for each ant, they apply the procedure described in Algorithm 3.7.

At the end of this procedure, the ant has visited all vertices of the graph (i.e., all job operations) according to the order in which set B was constructed. This sequence makes it possible to define a direction on all edges of the graph, thus defining an ordering of the operations on each machine. Colorni *et al*. then apply the longest route with minimal length algorithm to find the date at which the last job ends [ROY 64]. In addition to its applications to the traveling salesman and the plant problem, the ant algorithm was also adapted to graph coloring [COS 97] and quadratic assignment [MAN 99], among others. More details on ant colony algorithms can be found in [DOR 04].

Initialization
$f(s^*) \leftarrow$ *infinite;* (infinite = arbitrarily large value)
ncycles $\leftarrow 0$; (cycle counter)
best_cycle $\leftarrow 0$; (cycle leading to the best s^* solution found to date)
$\tau_1(..., i) := 1;\ \tau_2(..., i, j) := 1;\ \forall\ i = 1, ..., n\ \ \forall j = 1, ..., m$
Iterative process
while *no stopping criterion is met* **do**
  ncycles $\leftarrow$ *ncycles* + 1;
  for $q = 1$ **to** *nants* **do**
    for $k = 1$ **to** n **do**
      choose an object $i \in \{1, ..., n\} \setminus \{o(1), ..., o(k-1)\}$ *with probability* $p_{obj}(k, i)$;
      choose a feasible resource j for object i with probability $p_{res}(k, i, j)$;
      assign resource j to object i; $x_i := j$; $o(k) := i$;
    calculate the cost $f(s_q)$ *of solution* $s_q = (x_1, ..., x_n)$;
  set population p equal to the set $\{ s_1, s_2, ..., s_{nants} \}$
  set $s' = \arg\min \{f(s) \mid s \in p\}$;
  if $f(s') < f(s^*)$ **then** $s^* \leftarrow s'$ *and* *best_cycle* $\leftarrow$ *ncycles;*
  update traces τ_1 *and* τ_2

Algorithm 3.6. *An ant algorithm for a general assignment problem*

Progression of an ant
  Construct the graph $G_{jobshop}$ *where* α *represents the artificial vertex;*
  for every vertex x in $G_{jobshop}$ *set A(x) equal to the set {t | there is an arc from x to t or an edge between x and t };*
  set $A \leftarrow A(\alpha)$ *and* $B \leftarrow \varnothing$;
  set C equal to the vertex set of $G_{jobshop}$;
Iterative process
  while $C \neq \varnothing$ **do**
    choose a vertex $t \in A$ *according to a probability of transition;*
    set $A \leftarrow A \cup A(t) \setminus \{t\}$, $C \leftarrow C \setminus \{t\}$ *and* $B \leftarrow B \cup \{t\}$;

Algorithm 3.7. *Progression of an ant*

3.4.4. *The hybrid approach*

Evolutionary methods, and particularly genetic algorithms have been studied in depth since their very first developments early in the 1970s [HOL 75]. The many adaptations proposed in the literature fulfill the main weaknesses of traditional evolutionary methods where global performance is often far lower than that of a local search method such as tabu search or simulated annealing. In a more specific context, it is now understood that a simple genetic algorithm cannot provide good results when the set of solutions must satisfy many constraints. Grefenstette [GRE 87] has shown that it is possible to take advantage of the characteristics of the problem studied when defining all operators in a genetic algorithm.

Most of the innovations introduced in the evolutionary method field use concepts no longer related to natural evolution principles. Very interesting results were recently obtained by inserting a local search method into the individual adaptation phase of an evolutionary method. In what follows, we will refer to this new combined search method as a hybrid algorithm. The strength of a hybrid algorithm resides in the combination of these two fundamentally different search principles. The role of the local search method is to explore in depth a given region of the set X of solutions, whereas the evolutionary method introduces general conduct rules in order to guide the search through X. In this sense, the combination operator has a long term beneficial diversifying effect. To our knowledge, works by Glover [GLO 77], Grefenstette [GRE 87] and Mühlenbein *et al.* [MÜH 88] are the originators of hybrid algorithms. Each has introduced a simple descent method (see Algorithm 3.2) to increase the performance of an existing evolutionary method. Glover used a scatter search method and Grefenstette and Mühlenbein *et al*. used a genetic algorithm to solve integer programming and traveling salesman problems respectively. Simulated annealing and tabu search are improvements of the simple descent method. It is thus normal to use them within an evolutionary method to further increase its performance. Combination possibilities are numerous at this level [CAL 99, COS 95a, COS 95b, FAL 96, FLE 94, FLE 96b, MOS 93, REE 96].

In his thesis, Costa proposes two hybrid algorithms based on the two combination schemes described in Figure 3.6 [COS 95c]. The resulting algorithm is called an evolutionary tabu search or evolutionary descent algorithm depending on the local search method used. In the first case, the local search method plays the role of the mutation operator whereas in the second case it replaces the reproduction operator. This operator has been deleted to decrease premature convergence risks which are often high in a hybrid algorithm where the local search method is deterministic.

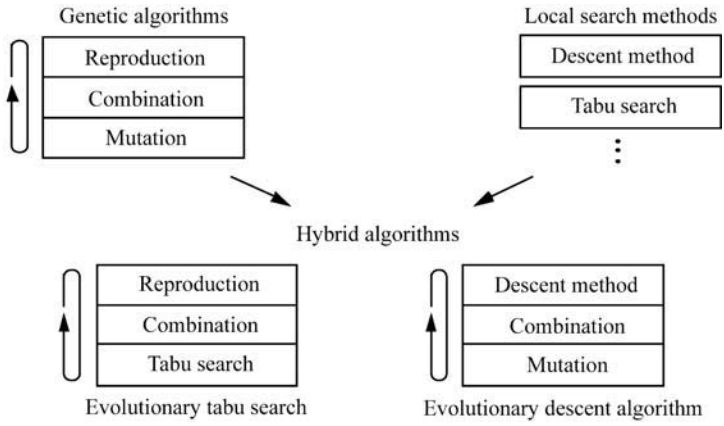


Figure 3.6. *Two hybrid schemes using a genetic algorithm*

The evolutionary tabu search has been applied to sports scheduling problems. Costa has shown that the performance of the algorithm is significantly better than the tabu search executed separately during a comparable time period. This type of behavior is the result of the complementarity which exists between tabu search and genetic algorithms. To be really efficient, tabu search requires significant modeling and parameter tuning efforts. Conversely, even though genetic algorithms are less efficient, they have the advantage of being robust and based on extremely simple rules. In addition, Costa presents an adaptation of the evolutionary descent algorithm to solve graph coloring problems [COS 95b]. The performance of the algorithm is much better than the genetic algorithm and the descent method used in it. Results obtained from a hybrid algorithm are usually of very high quality. Unfortunately, necessary computing times to reach a given quality solution can become prohibitive. After a comparison of several approaches in the solution of a range of quadratic assignment problems, Taillard [TAI 95] concludes that hybrid algorithms are among the most powerful but also the most time expensive. Choosing the right method largely depends on the time available for the solution of a specific problem. Nowadays research focuses on parallelizing hybrid algorithms in order to reduce computing time and solve larger problems.

3.5. An application example: job shop scheduling with tooling constraints

Production systems are constantly evolving, especially in the engineering industry. Rigid transfer lines are currently being replaced by cells or flexible workshops. These systems are made up of numerical control machines and can simultaneously produce different types of products. This versatility is essential in the

current manufacturing context. In fact, it is important to be able to produce different types of products quickly in order to satisfy client demand. This means that set-up times between two types of products must be very short. To reach this goal, optimal management of tool switches is necessary. Owing to this, one basic hypothesis of the scheduling problem in a workshop has become obsolete [BAK 74]: *set up times for operations are independent of the sequence and are included in processing times*.

The scheduling problem in a workshop considering tooling constraints is described in more detail below. The main elements are a series of machines and a collection of types of products to be produced on these machines. A process plan is associated with each product type and consists of a sequence of operations; each operation is characterized by a type of machine on which it must be performed, its processing time and the tools needed for each machine. The main hypotheses are as follows [BAK 74, WID 91a]:

– m different machines are continuously available (no breakdown and no maintenance operation are considered);

– a set of n type products is available for processing at time zero (each product type requires m operations and each operation is processed on a different machine);

– each machine has a tool magazine with a limited capacity;

– a machine can process only one product at a time;

– a product can only be produced by one machine at a time;

– individual operations are non-preemptive;

– each individual operation requires a number of tools, which never exceeds the capacity of the tool magazine;

– process plans are known in advance.

The goal is to find a sequence of operations for each machine to minimize the makespan which is defined as the maximum of the completion times of all operations.

This problem, *if tooling constraints are not taken into consideration*, is the well known job shop scheduling problem. It is NP-complete, as demonstrated by Lawler, *et al.* [LAW 89]. A reader interested in a good summary of methods implemented to solve the job shop scheduling problem without tooling constraints should refer to work by Blazewicz, Domschke and Pesch [BLA 96].

Returning now to our problem with tooling constraints, because of the limited capacity of the tool magazines on the machines, tool switches are unavoidable and

further complicate the problem of finding an optimal job shop scheduling [BLA 94]. When a *tool switch* happens, the machine's tool magazine is emptied of its tools, either partially or completely, and replenished with the tools necessary for the production of the next products in the planned sequence. The time required for this change is usually significant. To illustrate this, we will use the following example: we assume that products A, B and C must be manufactured (in that order) on a machine with a tool magazine having a capacity of 10. These products require respectively 4, 5 and 3 tools that are different from one another. A tool switch must therefore happen after the manufacturing of products A and B.

In certain cases, different types of products may require common tools. These tools should not be duplicated in the tool store, thus free space may be preserved. In our example, if A and B use 2 tools in common, it is possible to include tools required for the production of the three products in the store without exceeding its capacity (4 + 5 – 2 + 3 = 10). In these conditions, it is not necessary to execute a tool switch.

Before moving on to the resolution of the job shop scheduling problem with tooling constraints with a tabu search, we will discuss a traditional case model in order to draw some lessons from it.

3.5.1. *Traditional job shop modeling*

For clarity purposes, from now on, the traditional job shop scheduling problem will be noted as JP, while JPT will be the job shop scheduling problem with tooling constraints.

Let $\mathbf{O} = \{1, ..., n\}$ denote the total set of operations which are to be performed. For each operation $i \in \mathbf{O}$ we denote:

– P_i the product to which i belongs;

– M_i the machine on which i is processed;

– p_i the processing time of i;

– $PP(i)$ the operation preceding i in the process plan of P_i;

– $FP(i)$ the operation following i in the process plan of P_i .

A job shop scheduling problem can be represented as a graph theoretical problem [ROY 64]. For a given example of JP, the following graph G = (X, U, D) can be associated with it:

– $X = \{0, ..., n+1\} = \mathbf{O} \cup \{0, n+1\}$, where 0 and n + 1 are special vertices identifying the start and the completion of the schedule;

– $U = \{[i, j] \mid 1 \leq i, j \leq n\ , M_i = M_j \text{ and } P_i \neq P_j\}$

– $D = \{(i, j) \mid 1 \leq i, j \leq n \text{ and } j = FP(i)\}$
$\cup\{(0, i) \mid 1 \leq i \leq n \text{ and } i \text{ is the initial operation for product } P_i\}$
$\cup\{(i, n+1) \mid 1 \leq i \leq n \text{ and } i \text{ is the final operation for product } P_i\}$

With each arc $(i,j) \in D$ or edge $[i, j] \in U$ with $i \geq 1$, we associate a length representing the duration p_i of operation i. All arcs from vertex 0 have a length equal to zero. In other words, all arcs in D represent the different process plans. An orientation of the edges in U is called *feasible* if it does not create a circuit in G. A feasible orientation of the edge set U corresponds to a *feasible ordering* of the operations on the machines. The job shop scheduling problem is to find a feasible orientation of the edge set U in such a way that the longest route from 0 to $n + 1$ is of minimal length. Two operations requiring the same machine are defined as *adjacent* if the ending time of the first product is equal to the starting time of the second product on this same machine. An operation is called *critical* if it belongs to the longest route from 0 to $n + 1$. A *block* is a maximal sequence of adjacent operations which must be executed on a single machine and belonging to a longest route from 0 to $n + 1$.

The first adaptation of tabu search to the JP was developed by Taillard [TAI 94]. The set X of solutions is defined as the set of feasible orientations. Given a solution s in X, a neighbor solution $s' \in N(s)$ is obtained by permuting two consecutive critical operations using the same machine. When two consecutive critical operations i and j are permuted, operation j is introduced in the tabu list T: it is forbidden to permute j with the next operation on machine M_j during |T| iterations. The neighborhood structure N(s) used by Taillard has the following properties, demonstrated by van Laarhoven *et al.* [VAN 92]:

– if s is a feasible orientation, then all elements in N(s) are also feasible orientations;

– let **G** be the state space graph induced by X and N(s) (see section 3.4.2): for each solution $s \in X$, there is at least one path in **G** linking s to an optimal solution s^*.

A second tabu search adaptation to the JP was proposed by Dell'Amico and Trubian [DEL 93]. Their method seems to dominate Taillard's approach. The main difference is the definition of neighborhood N(s): for each operation i in a block, they consider as neighbors of s those solutions that can be generated by moving i to the first or last position of the block to which it belongs, if the corresponding ordering is feasible. If that is not the case, operation i is moved to the position inside

the block closest to the first or last position so that feasibility is preserved. Dell'Amico and Trubian have proved that this type of neighborhood leads to a state space graph **G** in which it is possible to reach an optimal solution starting from any initial solution [DEL 93].

Nowicki and Smutnicki [NOW 96] have developed a third adaptation of tabu search for the JP, which is based, once again, on a different definition of the neighborhood N(s). These three adaptations of tabu search for the JP unfortunately do not consider tooling constraints.

3.5.2. *Comparing both types of problems*

As mentioned in the introduction, the time required to switch tools on a machine is significant. The moment where the contents of a tool magazine are modified will be called a *switching instant*. It lasts $\alpha + \beta r$ time units, where α is a fixed time due to the removal of the tool magazine from the machine, β is a fixed time for each tool replacement and r is the total number of tools to be replaced by other tools. In addition, a complete set of tools is attached to each machine (in this way, two machines can use identical tools simultaneously).

With these basic hypotheses, a comparative analysis of JP and JPT problems was conducted to determine how ingredients used in solution methods for the JP could also be used for the JPT. The following observations were made [HER 95, HER 96]:

– an optimal ordering of the operations on the machines for the JP does not necessarily correspond to an optimal ordering for the JPT;

– given an ordering of the operations, minimizing the makespan is not necessarily achieved by minimizing the number of switching instants;

– given an ordering of the operations, a schedule of minimum time for the JP can easily be obtained by searching for a longest path in a graph; this problem can be solved in polynomial time (see for example the adaptation of Bellman's algorithm described in [TAI 94] or the $O(n)$ algorithm developed by Adams *et al.* [ADA 88]). For the JPT problem, the complexity of this problem is still open. In other words, given an ordering of the operations on the machines, we do not know how to take into account tooling constraints for minimizing the makespan;

– assuming that the number of switching instants is given for each machine, it is not always optimal to plan the switching instants to the earliest or latest;

– if we know the sequence of operations on each machine as well as the contents of the tool magazine, then a minimum time scheduling can be found by solving a longest path problem; in this case, switching instants are considered as additional

operations whose duration is known and the graph is constructed in the same way as for the JP.

As mentioned above, when solving the JP, neighborhood structures from Taillard and from Dell'Amico and Trubian generate a state space graph **G** in which it is possible to reach an optimal solution from any initial solution. This is unfortunately not the case with the JPT. In fact, by using these neighborhood structures, it is possible to "cycle" between a set of solutions while the optimal solution may be elsewhere (the state space graph **G** contains several connected components) [HER 96].

3.5.3. *Tool switching*

As previously mentioned, despite knowing operation sequences on machines, it is still difficult to determine the best way to take tooling constraints into consideration. One possibility is to use the heuristic method proposed in [HER 96]. It contains three phases which are summarized below.

In the first phase, all machines are treated independently, the aim being to evaluate the *number of switching instants*. The INSTANT algorithm determines a set of operations which immediately precede a switching instant. It considers the ordered set $\{o_1, ..., o_r\}$ of operations on a machine k ($1 \leq k \leq m$) and sequentially schedules the switching instants only when forced to. Thus, if a switching instant precedes an operation o_i ($1 < i \leq r$), this means that the set of tools required to complete o_i on k cannot be added to the tool magazine without exceeding its capacity. The number of switching instants determined by this algorithm is minimal. In the following, all considered schedules will have that number of switching instants.

The aim of the second heuristic phase is to determine the *contents of the tool magazines*. As in the first phase, all machines are treated independently. Since tool replacements can only occur during the switching instants, all operations between two consecutive switching instants can be considered as a unique operation requiring a known set of tools to be processed. Thus, given an ordered set of operations which must be processed on a machine k ($1 \leq k \leq m$), the objective is to minimize the total number of tool switches. It was proven by Tang and Denardo [TAN 88] that this problem can be solved in an exact way and in a reasonable amount of time. The REPLACEMENT algorithm uses the KTNS (keep tool needed soonest) policy proposed by Tang and Denardo [TAN 88]. This replacement policy has the following properties:

– at any given instant, no tool is introduced unless it is required for the next operation;

– if a tool must be inserted, the tools remaining (not taken out) are those needed the soonest.

Once the tool switching problem is solved on each machine, a solution to the JPT can be obtained by solving a longest path problem. In fact, all switching instants can be considered as additional operations with known duration, and a graph can then be constructed in the same way as for the JP. The two first heuristic phases consider the machines independently. In the third phase, all the machines are treated simultaneously, the aim being to *improve a solution* by scheduling a switching moment on a machine when no product is ready to be processed on it. An algorithm, called IMPROVE, is proposed in [HER 96] to improve the schedule of the switching instants.

3.5.4. *TOMATO algorithm*

We can now describe a tabu search algorithm for the job shop scheduling problem with tooling constraints.

Let *s'* be a neighbor solution of *s* obtained by moving an operation on a machine *k*. To evaluate *s'*, we first determine the operations on *k* which immediately precede a switching instant with the help of the INSTANT algorithm. We then apply the REPLACEMENT algorithm on *k* and calculate the makespan of *s'* by solving a longest path problem (see section 2.4.2). Once the decision is made to move from a current solution *s* to a neighbor solution $s' \in N(s)$, we try to improve the schedule of switching instants in *s'* by using the IMPROVE algorithm.

The tabu T list is a set of operations not authorized to be moved during a certain number of iterations. When an operation *i* is moved before or after an operation *j* on a machine *k*, operation *i* is introduced in the tabu list T. In addition, if *j* is an immediate predecessor or successor to *i*, operation *j* is also introduced in the tabu T: indeed, in that case, *s* could be obtained from *s'* by moving *j* before or after *i*. The length of the tabu list is randomly chosen at each iteration in the interval $[n, \lfloor 3n/2 \rfloor]$, following a uniform distribution.

The objective function *f* is the makespan (the time where the last job is completed). Finally, the iterative process ends when *nbmax* iterations are completed without improvement of $f(s^*)$. An algorithmic form of this tabu search adaptation to the job shop scheduling problem with tooling constraints is presented in Algorithm 3.8. The method was named TOMATO (for TOol MAnagement with Tabu Optimization) in [HER 96].

Initialization
find an initial solution for the JP;
let s be the schedule obtained by using the INSTANT and REPLACEMENT heuristics on each machine, and by solving a longest path problem;
improve s by using the IMPROVE procedure;
set nbiter ← 0, T ← ∅, s ← s and best_iter ← 0;*
Iterative process
while *(nbiter-best_iter<nbmax)* **do**
set nbiter ← nbiter+1;
evaluate each solution s' of N(s) by using the INSTANT and REPLACEMENT procedures for the machine on which an operation is moved, and by solving a longest path problem;
choose the best solution s' of N(s) so that f(s') < f(s) or s' is not tabu;*
improve s' by using the IMPROVE procedure;
update the list T of tabu movements;
set s ← s';
if *f(s) < f(s*)* ***then*** *set s* ← s and best_iter ← nbiter*

Algorithm 3.8. *The TOMATO algorithm*

TOMATO should only be applied to job shop scheduling problems in which the tool magazines are of relatively limited capacity, which is the case with small to medium-sized companies. When the capacity of the tool magazines increases and becomes far greater than the average number of tools necessary for the execution of an operation, the problem generally becomes similar to the job shop scheduling problem without tooling constraints. In this case, a solution can be obtained with the method proposed by Dell'Amico and Trubian, and (the few) switching instants can then be planned by using the INSTANT, REPLACEMENT and IMPROVE algorithms. TOMATO is a flexible method that can be used by a production manager who wants either a quick but not precise estimation of the makespan or an accurate solution. The desired trade-off between calculation time and solution quality can be established by adjusting the parameter *nbmax*. TOMATO can be used as a descent method by setting $nbmax = 0$. It was observed in [HER 96] that TOMATO provides in less than 1 second solutions whose value are approximately only 5% above the best known solution values.

3.6. Conclusion

Upon reaching the end of this chapter, a reader inevitably asks the following question: which is the best metaheuristic?

Unfortunately, a direct and specific answer is impossible: even though some similarities are obvious between these different approaches (e.g., the use of neighborhoods), they are different in sensitive areas (simulated annealing uses an energy function, tabu search handles a list of prohibited movements, genetic algorithms have crossover operators, etc.). Comparisons are difficult to make for two reasons: on the one hand, such a comparison requires a fine tuning of all parameters of all methods, and on the other hand, the quality of the solutions produced by a metaheuristic depends on the available computing time. However, regardless of the metaheuristic used, it is now a fact that all these algorithms are the only effective solution methods for many large size combinatorial optimization problems. The promising results of hybrid approaches bode well for vital progress in this field.

In conclusion, we mention that any reader wanting a deeper understanding of metaheuristics for general combinatorial optimization problems should see [REE 95, DRE 03]. Also, successful adaptations of metaheuristics for the solution of real problems are described in [IBA 05].

3.7. Bibliography

[AAR 89] AARTS E.H.L. and KORST J., *Simulated Annealing and Boltzmann Machines*, John Wiley, New York, 1989.

[ADA 88] ADAMS J., BALAS E. and ZAWACK D., "The shifting bottleneck procedure for job shop scheduling", *Management Science*, vol. 34, p. 391-401, 1988.

[BAK 74] BAKER K.R., *Introduction to Sequencing and Scheduling*, John Wiley, New York, 1974.

[BLA 94] BLAZEWICZ J. and FINKE G., "Scheduling with resource management in manufacturing systems", *European Journal of Operational Research,* vol. 76, p. 1-14, 1994.

[BLA 96] BLAZEWICZ J., DOMSCHKE W. and PESCH E., "The job shop scheduling problem: conventional and new solution techniques", *European Journal of Operational Research,* vol. 93, p. 1-33, 1996.

[CAL 99] CALEGARI P., CORAY G., HERTZ A., KOBLER D. and KUONEN P., "A taxonomy of evolutionary algorithms in combinatorial optimization", *Journal of Heuristics*, vol. 5, p. 145-158, 1999.

[CAR 88] CARLIER J. and CHRETIENNE P., *Problèmes d'ordonnancement*, Masson, 1988.

[CER 85] CERNY V., "Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm", *Journal of Optimization Theory and Applications*, vol. 45, p. 41-51, 1985.

[COL 88] COLLINS N.E. “Simulated annealing: an annotated bibliography”, *American Journal of Mathematical and Management Sciences*, vol. 8, p. 209-307, 1988.

[COL 91] COLORNI A., DORIGO M. and MANIEZZO V., “Distributed optimization by ant colonies”, *First European Conference on Artificial Life*, Paris, p 134-142, 1991.

[COL 92] COLORNI A., DORIGO M. and MANIEZZO V., “An investigation of some properties of an ant algorithm”, *Second Conference on Parallel Problem Solving from Nature*, Brussels, p. 509-520, 1992.

[COL 94] COLORNI A., DORIGO M., MANIEZZO V. and TRUBIAN M., “Ant system for job shop scheduling”, *Belgian Journal of Operations Research, Statistics and Computer Science*, vol. 34, p. 39-53, 1994.

[CON 88] Committee on the Next Decade of Operations Research (CONDOR), “Operations research: the next decade”, *Operations Research* , vol. 36, p. 619-637, 1988.

[COS 95a] COSTA D., “An evolutionary tabu search algorithm and the NHL scheduling problem”, *INFOR*, vol. 33, p. 161-178, 1995.

[COS 95b] COSTA D., HERTZ A. and DUBUIS O., “Embedding of sequential procedures within an evolutionary algorithm for coloring problems in graphs”, *Journal of Heuristics*, vol. 1, p. 105-128, 1995.

[COS 95c] COSTA D., Méthodes de résolution constructives, séquentielles et évolutives pour des problèmes d’affectation sous contraintes, Thesis no. 1411, Department of Mathematics, Ecole Polytechnique Fédérale de Lausanne, 1995.

[COS 97] COSTA D. and HERTZ A., “Ants can colour graphs”, *Journal of the Operational Research Society*, vol. 48, p. 295-305, 1997.

[CUN 97] CUNG V.-D., MAUTOR T., MICHELON P. and TAVARES A., “A Scatter search based approach for the quadratic assignment problem”, *IEEE International Conference on Evolutionary Computation*, p. 165-170, 1997.

[DEL 93] DELL’AMICO M. and TRUBIAN M., “Applying tabu search to the job-shop scheduling problem”, *Annals of Operations Research*, vol. 41, p. 231-252, 1993.

[DOR 96] DORIGO M., MANIEZZO V. and COLORNI A., “The ant system: optimization by a colony of cooperating agents”, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 26, p. 29-41, 1996.

[DOR 04] DORIGO M. and STÜTZLE T., *Ant Colony Optimization*, MIT Press, Cambridge, 2004.

[DRE 03] DRÉO J., PÉTROWSKI A., SIARRY P. and TAILLARD É., *Métaheuristiques pour l’optimisation difficile*, Éditions Eyrolles, Paris, 2003.

[DUE 90] DUECK G. and SCHEUER T., “Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing”, *Journal of Computational Physics*, vol. 90, p. 161-175, 1990.

[DUE 93] DUECK G., “New optimization heuristics: the great deluge algorithm and the record-to-record travel”, *Journal of Computational Physics*, vol. 104, p. 86-92, 1993.

[EGL 90] EGLESE R.W., "Simulated annealing: a tool for operational research", *European Journal of Operational Research*, vol. 46, p. 271-281, 1990.

[FAI 92] FAIGLE U. and KERN W., "Some convergence results for probabilistic tabu search", *ORSA Journal on Computing*, vol. 4, p. 32-37, 1992.

[FAL 96] FALKENAUER E., "A hybrid grouping genetic algorithm for bin packing", *Journal of Heuristics*, vol. 2, p. 5-30, 1996.

[FER 96] FERLAND J.A., HERTZ A. and LAVOIE A., "An object oriented methodology for solving assignment type problems with neighborhood search techniques", *Operations Research*, vol. 44, p. 347-359, 1996.

[FLE 94] FLEURENT C., Algorithmes génétiques hybrides pour l'optimisation combinatoire, Thesis, Department of Information Technology and Operations Research, University of Montreal, 1994.

[FLE 96a] FLEURENT C., GLOVER F., MICHELON P. and VALLI Z., "A scatter search approach for unconstrained continuous optimization", *IEEE International Conference on Evolutionary Computation*, p. 643-648, 1996.

[FLE 96b] FLEURENT C. and FERLAND J.A., "Genetic and hybrid algorithms for graph coloring", *Annals of Operations Research*, vol. 63, p. 437-461, 1996.

[GAL 06] GALINIER P. and HERTZ A., "A Survey of Local Search Methods for Graph Coloring", *Computers and Operations Research*, vol. 33, p. 2547-2562, 2006.

[GAR 79] GAREY M.R. and JOHNSON D.S., *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, 1979.

[GEM 84] GEMAN S. and GEMAN D., "Stochastic relaxation, Gibbs distributions, and bayesian restoration of images", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, p. 721-741, 1984.

[GLA 96] GLASS C.A. and POTTS C.N., "A comparison of local search methods for flow shop scheduling", *Annals of Operations Research*, vol. 63, p. 489-509, 1996.

[GLO 77] GLOVER F., "Heuristics for integer programming using surrogate constraints", *Decision Sciences*, vol. 8, p. 156-166, 1977.

[GLO 86] GLOVER F., "Future paths for integer programming and links to artificial intelligence", *Computers and Operations Research*, vol. 13, p. 533-549, 1986.

[GLO 89] GLOVER F., "Tabu search: Part I", *ORSA Journal on Computing*, vol. 1, p. 190-206, 1989.

[GLO 90] GLOVER F., "Tabu search: Part II", *ORSA Journal on Computing*, vol. 2, p. 4-32, 1990.

[GLO 93a] GLOVER F., TAILLARD E., LAGUNA M. and DE WERRA D. (Eds.), "Tabu search", *Annals of Operations Research*, vol. 41, 1993.

[GLO 93b] GLOVER F., TAILLARD E. and DE WERRA D., "A user's guide to tabu search", *Annals of Operations Research,* vol. 41, p. 3-28, 1993.

[GLO 94] GLOVER F., "Genetic algorithms and scatter search: unsuspected potentials", *Statistics and Computing*, vol. 4, p. 131-140, 1994.

[GLO 95] GLOVER F., "Scatter search and star-paths: beyond the genetic metaphor", *OR Spektrum*, vol. 17, p. 125-138, 1995.

[GLO 97] GLOVER F. and LAGUNA M., *Tabu Search*, Kluwer Academic Publishers, Norwell, 1997.

[GOL 87] GOLDBERG D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley, 1987.

[GON 85] GONDRAN M. and MINOUX M., *Graphes et algorithmes*, Eyrolles, Paris, 1985.

[GRE 87] GREFENSTETTE J.J., "Incorporating problem specific knowledge into genetic algorithms", in *Genetic Algorithms and Simulated Annealing*, Davis. L. (ed.), Morgan Kaufmann Publishers, p. 42-60, 1987.

[HAJ 88] HAJEK B., "Cooling schedules for optimal annealing", *Mathematics of Operations Research*, vol. 13, p. 311-329, 1988.

[HAN 86] HANSEN P., "The steepest ascent mildest descent heuristic for combinatorial programming", *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy, 1986.

[HER 95] HERTZ A. and WIDMER M., "La méthode TABOU appliquée aux problèmes d'ordonnancement", *RAIRO/Automatique, Productique, Informatique Industrielle*, vol. 29, p. 353-378, 1995.

[HER 96] HERTZ A. and WIDMER M., "An improved tabu search approach for solving the job shop scheduling problem with tooling constraints", *Discrete Applied Mathematics*, vol. 65, p. 319-346, 1996.

[HER 97] HERTZ A., TAILLARD E. and DE WERRA D., "Tabu search", in *Local Search in Combinatorial Optimization,* E. Aarts and J.K. Lenstra (eds.), John Wiley, 1997.

[HER 03] HERTZ A. and WIDMER M., "Guidelines for the use of meta-heuristics in combinatorial optimization", *European Journal of Operational Research*, vol. 151, p. 247-252, 2003.

[HOL 75] HOLLAND J.H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[IBA 05] IBARAKI T., NONOBE K. and YAGIURA M., *Metaheuristics: Progress as Real Problem Solvers*, Springer, 2005.

[KEL 96] KELLY J., RANGASWAMY B. and XU J., "A scatter search-based learning algorithm for neural network training", *Journal of Heuristics*, vol. 2, p. 129-146, 1996.

[KIR 83] KIRKPATRICK S., GELATT C.D. Jr. and VECCHI M.P., "Optimization by simulated annealing", *Science*, vol. 220, p. 671-680, 1983.

[LAW 89] LAWLER E.L., LENSTRA J.K., RINNOOY KAN A.H.G. and SHMOYS D.B., "Sequencing and scheduling: algorithms and complexity", Chapter 9 in *Logistics of Production and Inventory*, Graves S.C., Rinnooy Kan A.H.G. and Zipkin P.H. (Eds.), Elsevier Science Publishers, p. 445-522, 1993.

[MAN 99] MANIEZZO V. and COLORNI A., "The ant system applied to the quadratic assignment problem", *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, p. 769-778, 1999.

[MET 53] METROPOLIS N., ROSENBLUTH A., ROSENBLUTH M., TELLER A. and TELLER E., "Equation of state calculations by fast computing machines", *Journal of Chemical Physics*, vol. 21, p. 1087-1092, 1953.

[MOS 93] MOSCATO P., "An introduction to population approaches for optimization and hierarchical objective functions: a discussion on the role of tabu search", *Annals of Operations Research*, vol. 41, p. 85-121, 1993.

[MÜH 88] MÜHLENBEIN H., GORGES-SCHLEUTER M. and KRÄMER O., "Evolution algorithms in combinatorial optimization", *Parallel Computing*, vol. 7, 65-85, 1988.

[MÜL 81] MÜLLER-MERBACH H., "Heuristics and their design: a survey", *European Journal of Operational Research*, vol. 8, p. 1-23, 1981.

[NAW 83] NAWAZ M., ENSCORE JR. E.E. and HAM I., "A heuristic algorithm for the m-machine, n-job flow shop sequencing problem", *Omega*, vol. 11, p. 91-95, 1983.

[NIC 71] NICHOLSON T.A.J., *Optimization in Industry*, *vol. 1: Optimization Techniques*, Longman Press, London, 1971.

[NOW 96] NOWICKI E. and SMUTNICKI C., "A fast tabu search algorithm for the job-shop problem", *Management Science*, vol. 42, p. 797-813, 1996.

[OSM 94] OSMAN I.H. and CHRISTOFIDES N., "Capacitated clustering problems by hybrid simulated annealing and tabu search", *International Transactions in Operational Research*, vol. 1, p. 317-336, 1994.

[RAD 94] RADCLIFFE N.J. and SURRY P.D., "Formal memetic algorithms", *Lecture Notes in Computer Science*, vol. 865, p. 1-16, 1994

[REE 93] REEVES C.R., *Modern Heuristic Techniques for Combinatorial Optimization*, Blackwell Scientific Publications, Oxford, 1993.

[REE 95] REEVES C.R., *Modern Heuristic Techniques for Combinatorial Problems*, McGraw-Hill, 1995.

[REE 96] REEVES C., "Hybrid genetic algorithms for bin-packing and related problems", *Annals of Operations Research*, vol. 63, p. 371-396, 1996.

[ROY 64] ROY B. and SUSSMANN B., Les problèmes d'ordonnancement avec contraintes disjonctives, Technical report, SEMA, Montrouge, 1964.

[RUT 89] RUTENBAR R. A., "Simulated annealing algorithms: an overview", *IEEE Circuits and Devices Magazine*, vol. 5, p. 19-26, 1989.

[SIL 80] SILVER E.A., VIDAL R.V. and DE WERRA D., "A tutorial on heuristic methods", *European Journal of Operational Research*, vol. 5, p. 153-162, 1980.

[SIN 93] SINCLAIR M., "Comparison of the performance of modern heuristics for combinatorial optimization on real data", *Computers and Operations Research*, vol. 20, p. 687-695, 1993.

[SOR 94] SORIANO P., Applications de la méthode de recherche avec tabous à divers problèmes d'optimisation combinatoire, Thesis, Department of Information Technology and Operations Research, University of Montreal, 1994.

[TAI 93] TAILLARD E., Recherches itératives dirigées parallèles, Thesis no. 1153, Department of Mathematics, Ecole Polytechnique Fédérale de Lausanne, 1993.

[TAI 94] TAILLARD E., "Parallel taboo search technique for the job shop scheduling problem", *ORSA Journal on Computing*, vol. 6, p. 108-117, 1994.

[TAI 95] TAILLARD E., "Comparison of iterative searches for the quadratic assignment problem", *Location Science*, vol. 3, p. 87-105, 1995.

[TAN 88] TANG C.S. and DENARDO V., "Models arising from a flexible manufacturing machine, Part I: Minimization of the number of tool switches, Part II: Minimization of the number of switching moments", *Operations Research*, vol. 36, p. 778-784, 1988.

[THE 94] THERAULAZ G., BONABEAU E., GOSS S. and DENEUBOURG J.L., "L'intelligence collective", *Pour la Science*, no. 128, 1994.

[VAN 87] VAN LAARHOVEN P.J.M. and AARTS E.H.L., *Simulated Annealing: Theory and Applications*, D. Reidel Publishing Company, Dordrecht, 1987.

[VAN 92] VAN LAARHOVEN P.J.M., AARTS E.H.L. and LENSTRA J.K., "Job shop scheduling by simulated annealing", *Operations Research*, vol. 40, p. 113-125, 1992.

[VID 93] VIDAL R.V. (ed.), *Applied Simulated Annealing*, Lecture Notes in Economics and Mathematical Systems 396, Springer-Verlag, Berlin, 1993.

[WER 89] DE WERRA D. and HERTZ A., "Tabu search techniques: a tutorial and an application to neural networks", *OR Spektrum*, vol. 11, p. 131-141, 1989.

[WID 91a] WIDMER M., "Job shop scheduling with tooling constraints: a tabu search approach", *Journal of the Operational Research Society*, vol. 42, p. 75-82, 1991.

[WID 91b] WIDMER M., *Modèles mathématiques pour une gestion efficace des ateliers flexibles*, Presses Polytechniques and Universitaires Romandes, 1991.

[WID 98] WIDMER M., Organisation industrielle: le réel apport des mathématiques, Habilitation thesis, Freiburg University, 1998.

[ZAN 89] ZANAKIS H.S., EVANS J.R. and VAZACOPOULOS A.A., "Heuristic methods and applications: a categorized survey", *European Journal of Operational Research*, vol. 43, p. 88-110, 1989.

Chapter 4

Genetic Algorithms and Scheduling

4.1. Introduction

4.1.1. *Origin of genetic algorithms*

Genetic algorithms (GA) are methods similar to simulated annealing (see Chapter 3) (they contain no inherent principle to assert that the best solution obtained in a finite time is optimal). They have also been created by analogy with natural phenomena. In this case, the idea is to simulate the natural evolution of organisms (individuals), generation upon generation, considering heredity and survival. In a population of individuals, generally the fittest, those best adapted to the environment, survive and ensure descendants. In addition, it is assumed that qualities and faults can be stochastically inherited from parents. Such algorithms were developed in 1950 by biologists using computers to simulate organism evolution. Late in the 1960s, John Holland [HOL 75] and his team, and later numerous other researchers [DAV 91, GOL 89], adapted these algorithms for designing solution methods for optimization problems, by developing an analogy between an individual in a population and a problem solution within a set of solutions.

4.1.2. *General principles of genetic algorithms*

In a population, an individual is characterized by his genetic footprint called a *genome* (a set of chromosomes). A genetic footprint is associated with each

---

Chapter written by Marie-Claude PORTMANN and Antony VIGNIER.

individual integrated into the initial population; it could be an elaborated footprint built by specific modules or a footprint obtained in a random way. The footprint of new individuals is further obtained by a recombination of footprints from parents, a genetic operation called *crossover* or by modifying the footprint of an individual, a genetic operation called *mutation*. The crossover corresponds to the sexual reproduction of individuals in a population with consideration for heredity. In this way, when two individuals considered fit enough are crossed, they will create a new individual who will also have good chances of being fit and resist natural selection, which is less the case with unfit individuals. Mutation represents modifications to the genetic footprint which may occur with individuals and avoids population degeneration. The genetic crossover operator increases population size. The population size is maintained at a reasonable finite size with the existence of a birth-and-death process according to rules depending on their fitness.

By analogy, in a genetic algorithm, an individual (a solution) is characterized by a data structure representing its genetic footprint, which we continue to call a genome or chromosome if the genome is reduced to a chromosome. The genome contains a set of codes called genes which in certain cases need to be interpreted to find the corresponding solution. In this case, the result of this interpretation is called a *phenotype*. With the help of these genomes, a genetic algorithm explores the space of solutions for the problem concerned in order to extract the best possible solution, and preferably the optimal solution. It is not necessary to explore the whole space; exploring a sub-space containing an optimal solution is sufficient. This is what is called a *dominant subset*. Coding and genetic operators must be designed carefully accordingly to the potential existence of a dominant subset. In this chapter, we will draw attention as much as possible to the dominance conditions for the problems addressed.

If we are trying to maximize a function, the fitness of an individual can be measured by a value which varies in the same direction as the objective function of the problem concerned, otherwise in the opposite direction if the objective function needs to be minimized. The greater the fitness, the better chances an individual will have to be chosen by a selection using a roulette technique (an individual's probability of being selected = his fitness divided by the sum of all individuals' fitness in a population). For each problem to be solved, we must determine how to obtain the fitness of an individual from the objective function. As a value of fitness, we can use the value of the objective function (if it must be maximized) or an upper bound of the objective function for the whole population less the value of the objective function (if it must be minimized). However, we can also, in the case of a maximization, for example, sort individuals by ascending objective function and then consecutively give them the fitness value: a, $a + \varepsilon$, $a + 2\varepsilon$, $a + 3\varepsilon$, etc.

Genetic crossover and mutation operators are algorithms acting on chromosomes associated with individuals. They are dedicated to exploring the space of problem solutions.

The population's birth-and-death process can be carried out in different ways. In certain cases, it is assumed that two consecutive generations do not live in the same period and thus that the parents are dead when children are alive (this is the case in reality for butterfly populations for example). In other cases, there is overlap between generations; grandparents can live at the same time as their grandchildren whereas the parents may have died prematurely. In both cases, based on an initial population and after numerous iterations where the population evolves, we reach a population where individuals are all very fit (converging to good quality population). In other words, it is a set of "good" solutions for the problem that concerns us. For the same problem, there are several genetic algorithm variations based on:

– coding used to represent the solutions in the form of genomes;

– the way to create an initial population;

– the way to transform an objective function into a fitness;

– selection mode for reproducing chromosomes (in general, the "better" an individual is, the better his chances for being chosen as a reproductive element);

– design of genetic crossover and mutation operators;

– the manner in which the population evolves throughout iterations, i.e. definition of the birth-and-death process.

The method of building a genetic algorithm, regardless of the problem to address, will be summarized. Throughout the rest of the chapter, emphasis will be made on quality of coding and genetic operators for designing specific and efficient genetic algorithms for given scheduling problems.

The simplest schema for a genetic algorithm [GOL 89] is illustrated in the following diagram (Algorithm 4.1). This is a schema of consecutive generations with no overlap between consecutive populations (nevertheless, since a pair of parents can simply be copies without crossover or mutation into the next population, we may find identical individuals in the previous and new populations).

a- INITIALIZATION:	Generate an initial P_o population of Q individuals.
b- EVALUATION:	Evaluate the "fitness" of any individual in population P_{k-1}.
c- SELECTION:	Select $Q/2$ individual couples in population P_{k-1}.
d- CROSSOVER:	With probability p_c any individual couple is replaced by a new individual couple obtained by applying a genetic crossover operator, with probability $1 - p_c$ any individual couple is retained (for example $p_c = 0.75$, i.e. rather large).
e- MUTATION:	With probability p_m any individual of P_k is modified by a mutation, with a probability $1 - p_m$ the individual is retained (for example $p_m = 0.2$, or rather small).
f-	Population P_k obtained replaces population P_{k-1}.
g- STOP?:	Go back to b- until a given number of iterations is reached.

Algorithm 4.1. *Simple Goldberg schema for genetic algorithms (SGA)*

To implement the Goldberg schema, we must choose:

– population size: Q,

– probabilities p_c and p_m,

– a number of generations before stopping.

However, we can also use the schema with population recovery proposed by [DAV 91] and several other variations with hybridization briefly described in section 4.5.

4.1.3. *Schema theorem*

This theorem applies to the simple Goldberg schema when gene coding is binary. Holland's schema theorem will not be presented in its totality here. We will only summarize its consequences. The schema theorem justifies the convergence (still ascertained, when respecting the genetic algorithm hypotheses) of genetic algorithms to a population of good individuals. It also shows the importance of the position of the genes when they are dependent (for example, if for a pair of given genes, solutions are better when genes have opposite values regardless of the value of these two genes, it is good that their positions increase the probability of taking their values in only one of both parents when applying a crossover operator). This idea, corresponding to the simplest genetic algorithm case for which the schema theorem is shown, can be extended to more complex coding and in this case, the consequence for the potential generalization of the schema theorem is as follows:

"What must be kept from the good characteristics of both parents in order to build children that are as good as possible?" Research of good crossover for the different problems addressed is presented in this chapter.

4.1.4. *Chapter presentation*

The structure of this chapter is based on an increasing complexity of scheduling problems. The complexity is defined here by the difficulty of developing genetic algorithms to solve these problems, which in fact constitutes a typology of these problems with regards to genetic algorithms. The first problems discussed are problems with one machine where coding is only permutation coding. It is illustrated with examples and shows that any coding and especially all crossover operators are not equivalent in developing efficient genetic algorithms. The case of job shop scheduling is then considered in order to show that permutation coding is not adequate for this type of problem and that they either need to be adapted or changed. The case with multiple parallel resources organized in a hybrid flow shop is introduced. We begin with a discussion of a simple case with only one stage (parallel machine problem). We then generalize to a case with multiple stages. A few combinations of operations research methods with genetic algorithms to try and improve their respective efficiency is presented briefly at the end of this chapter.

4.2. One-machine problems

4.2.1. *Example 1: total time and setup times*

Problem description

The problem involved is an extension of the basic one-machine problem. In the basic one-machine problem:

– the machine is always available,

– all operations are available at moment 0,

– setup times are independent of the sequence and included in operation time,

– all operation characteristics (times, due dates, etc.) are known at the moment when we address the problem.

In this extension, we presume that setup times depend on the sequence of operations: between operation i and operation j for example, tools for the numerically controlled machine must be changed and this takes s_{ij}. The criterion is minimizing total scheduling time, or what is equivalent, minimizing the total setup

times. We presume that we know the state of the machine at time 0 and consequently setup time s_{0i} between a fictitious operation numbered 0 of zero duration placed at moment 0 and any other operation i placed first. In the end, the machine is left in the last state of the last operation.

Preliminary analysis

In the traditional schema of genetic algorithms, we presume that the probability of an individual being chosen to procreate is higher if his quality (or fitness) is better and we also presume, as with natural reproduction, that two good individuals copulating will have a higher probability of producing children of good quality, maybe even exceptional children. If we want to reproduce these natural reproduction properties, we must thus give more chances to good individuals to procreate (this is done by pair selection in the global schema), as well as developing crossovers with a high probability of producing very good children when good individuals marry each other. In order to carry out such crossovers, we start with a preliminary analysis phase in each example to identify the good characteristics for each individual to attempt their reproduction in children.

In a genetic algorithm, when crossovers are not carefully developed, obtaining good quality children is left to chance and the genetic algorithms become unguided random explorations. Generally, in this case the authors feel the need to reverse the roles of genetic operators by asking crossover operators to create diversity instead of creating quality and mutation operators to improve individuals with improvement by neighborhood methods for example, instead of creating diversity. These authors continue to call their approach a "genetic algorithm", although they have strayed from the original spirit of genetic algorithms which is respected throughout this chapter.

Concerning the problem addressed in this section, a solution is described by the sequence of operations on the machine (see Figure 4.1 illustrating a solution in the form of a Gantt chart). A solution is right if setup times used to make up this solution are short.

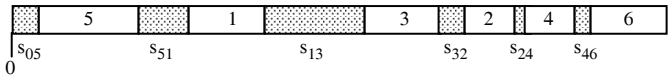


Figure 4.1. *Gantt chart of a one-machine problem with setup times*

When an individual "father" and an individual "mother" are good, it is because setup times are short. In order to create children, setup times must be those present in the father or the mother or the shortest possible. The idea here is to select couples *(i, j)* corresponding to an immediate set of operations on machines by giving

preference to couples *(i, j)* which correspond to short setup times. These are the good schemas that we should attempt to retain during crossovers.

Possible solution codings

Many codings can be used to represent a permutation of *n* elements corresponding to the sequence of *n* operations on the machine. Only two codings are presented here and their qualities and faults are analyzed.

The first coding is called "permutation". It simply consists of placing in a vector the number of operations in the sequence where they are placed on the machine. For Figure 4.1, the permutation vector is given in Table 4.1.

Position in the sequence	1	2	3	4	5	6
Permutation	5	1	3	2	4	6

Table 4.1. *The permutation vector*

The second coding is called "rank". It consists of giving the relative position of each operation on a machine in a vector. For Figure 4.1, the rank vector is given in Table 4.2.

Operation number	1	2	3	4	5	6
Rank	2	4	3	5	1	6

Table 4.2. *The rank vector*

Generation of initial population

In order to generate an individual, we must determine the sequence of operations and then transcribe this sequence, either in the form of permutation coding or in the form of rank coding.

The first method for obtaining individuals for the initial population is to generate coding in a completely random manner. We generate a regular sequence with probability 1/n! and we place the sequence obtained in the permutation vector or in the rank vector. This will generate a very diversified sub-population.

The second method for obtaining individuals for the initial population is to use similar resolution methods with or without the introduction of random decisions in the methods. This generates a generally good quality sub-population. Since the

problem concerned here is a specific case of the traveling salesman problem, we can use polynomial heuristics designed for this problem (for example, closest visited city or closest insertion, the simplest and most well known).

Crossover operators

Only two crossover operators are described (we can find many more in the literature; see for example [POR 96]). They are presented with permutation coding with emphasis on quality. Their effect on rank coding is then examined. The most traditional operator used in genetic algorithms is the one-point operator consisting of copying the father's beginning until the randomly selected point of crossover and the end of the mother from the crossover point for the son (for the daughter, switch "father" and "mother"). Unfortunately, this very simple crossover is not suitable for the permutation vector or the rank vector as is illustrated in Table 4.3, since the results obtained may no longer be permutations of 1, 2, …, *n* (for example, value 6 is absent and value 4 is duplicated in the son's encoding). It is therefore important to design specific crossovers for the codings representing permutations.

Father coding	2	4	3	5	1	6
Mother coding	6	2	5	3	1	4
Son coding	2	4	5	3	1	4
Daughter coding	6	2	3	5	1	6

Table 4.3. *Simple one-point crossover for permutation or rank vectors*

The first crossover operator presented here is the simplest version of the sequence operator proposed in [DAV 85]. It is represented by the acronym 1X (where 1 means that it is a one-point crossover, X symbolizes crossovers. If we were to be rigorous, it should be represented as 1OX for one-point order crossover). As with the simple one-point operator, it begins with a copy of the father's beginning (resp. of the mother) and it ends the chromosome with missing sequenced digital values sorted in the order of the mother (resp. father). By switching "beginning" and "end" in this crossover's description, we obtain another son and another daughter based on a similar principle, which would, for example, make it possible to keep the two best children obtained, thus improving the quality of this crossover. Table 4.4 illustrates the 1X crossover with the same example as the one from Table 4.3.

We now examine the children created. We may presume that if the father and the mother were good quality, it is because corresponding setup times were relatively short. Working with the permutation vector, this is translated by the hypothesis that

$s02$, $s24$, $s43$, $s35$, $s51$, $s16$, $s06$, $s62$, $s25$, $s53$, $s31$, $s14$ were relatively short compared to the rest of setup times. It would then be interesting to examine how these setup times are kept in the children created. We find four in son 1, six in daughter 1, six in son 2 which is identical to the father, and four in daughter 2.

Father coding	2	4	3	5	1	6
Mother coding	6	2	5	3	1	4
Son 1 coding	2	4	6	5	3	1
Daughter 1 coding	6	2	4	3	5	1
Son 2 coding	2	4	3	5	1	6
Daughter 2 coding	2	6	5	3	1	4

Table 4.4. *1X sequence crossover*

Father permutation	5	1	3	2	4	6
Mother permutation	5	2	4	6	3	1
Son 1 permutation	6	1	5	2	4	3
Daughter 1 permutation	6	2	4	3	5	1
Son 2 permutation	5	1	3	2	4	6
Daughter 2 permutation	5	1	4	6	3	2

Table 4.5. *Equivalent permutations to Table 4.4*

Table 4.5 shows the equivalent of permutation coding for individuals from Table 4.4 presumably defined by a rank coding.

If we assume the coding corresponds to the ranks (see Table 4.5) then we discover two setups from parents in son 1 and daughter 1, six in son 2 which is identical to the father and five in daughter 2. We see in the example that globally we find less subsequent operation pairs from parents if the 1X crossover is applied to rank coding instead of permutation coding and this is explained by the fact that 1X better retains consecutive operation sub-sequences for permutation than for rank coding, where what is retained or discovered is carried out in a much more random way.

Another operator was designed specifically to retain the routes between cities (edges $\{i, j\}$ or $\{j, i\}$) for the symmetric traveling salesman (distance from i to j = distance from j to i) by [WHI 89]. This is the edge recombination crossover (ERX).

We can easily extend it to pair conservation in the case of asymmetric problems. This extension is presented here. We explain it by using permutation coding in the example in Table 4.4. The first phase consists of building the table of pairs belonging to both parents. It is represented in Table 4.6 in the form of a first element successors list of each pair.

Entries in the table	0	1	2	3	4	5	6
List of successors	2	4	4	1	3	1	2
	6	6	5	5	/	3	/

Table 4.6. *Table of pairs*

In our particular case, we start with fictitious 0 operation. This is the current operation. Then Algorithm 4.2 is executed.

while there remains an operation not placed **do**

Place the current operation;

Among the non-placed successors from the current operation, arbitrarily choose one among those with the least amount of successors (*) not yet placed or arbitrarily choose a non-placed operation if the current operation has no non-placed successors;

The chosen operation becomes the current operation. **End while**.

Algorithm 4.2. *Pair recombination crossover*

The result when using Table 4.6 is:

0	6	2	4	3	1	5

We retain 5 out of 6 arcs which could have been retained. By replacing successors by predecessors where there is a star (*) in the algorithm, we observe better results. Based on the different random choices illustrated by numbers in boxes, we obtain the following results:

0	2	4	3	5	1	6
0	2	5	1	4	3	6
0	2	5	3	1	4	6
0	2	5	3	1	6	4

0	6	2	4	3	1	5
0	6	2	5	1	4	3
0	6	2	5	3	1	4

4 times five arcs are retained and 3 times six arcs are retained. We retain many arcs which are probably not very costly; on the other hand, the quality of added arcs is random, and because of this, some children will probably be good and others mediocre (they disappear rather quickly with the birth-and-death process).

Mutation operators

The mutation operator is intended for the creation of diversity in the population. It must lightly modify the permutation involved while retaining a permutation. We must change at least two pairs. We can draw a position i ($< n$) and switch values of positions i and $i + 1$, or draw a position i and a position j ($\neq i$) and switch values of positions i and j, or even move the value corresponding to position i within permutation forward or backward.

4.2.2. *Example 2: sum of weighted tardiness*

Problem description

We focus on the basic one-machine problem (see its description in section 4.2.1). The criteria to optimize is the sum of weighted tardiness in relation to due dates, in other words, any operation i has a due date d_i and is considered late if it ends at time C_i where C_i is strictly greater than d_i. An operation i not finished on time leads to a penalty equal to its tardiness in relation to due date, $\max(0, C_i - d_i)$, multiplied by a penalty associated with operation i: w_i.

Preliminary analysis

A solution to this problem can be described again by the sequence of operations on the machine (see Figure 4.2, where lines model the tardiness and the number of lines denote the tardiness penalty value). Intuitively, a solution is "good" (it corresponds to a low value of the sum of weighted tardiness criteria) if most operations are finished on time and if the operations which are the latest do not correspond to high penalty operations.

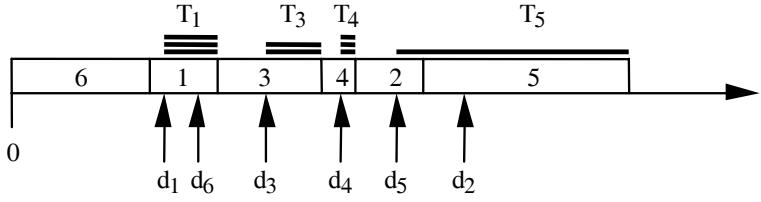


Figure 4.2. *Gantt chart of a one-machine problem with sum of weighted tardiness*

A "father" individual and a "mother" individual are considered good when they place short due date and/or high tardiness penalty operations early in the sequence (and in certain cases, the shortest operations before the longest operations). Relative positions between operations, or partial sequences, are what should be kept from good "father" and "mother" individuals as they constitute the good schemas to retain for crossovers.

Possible solution codings

As with the previous example, the permutation vector is used. It directly describes the sequence of operations as it appears in the Gantt chart. We can also use matrix coding called "permutation matrix" to work on total sequences (with binary coding), as well as on partial sequences (with ternary coding). Ternary coding is the only one presented; it will be used for other scheduling problems more complex than the one-machine problem. Table 4.7 illustrates this coding for permutation of Figure 4.2.

	1	2	3	4	5	6
1	0	1	1	1	1	-1
2	-1	0	-1	-1	1	-1
3	-1	1	0	1	1	-1
4	-1	1	-1	0	1	-1
5	-1	-1	-1	-1	0	-1
6	1	1	1	1	1	0

Table 4.7. *MT permutation matrix*

This permutation matrix is defined as MT(i, i) = 0 and MT(i, j) = 1 if i precedes j in the permutation, otherwise –1. It has specific properties. It is transitive: MT(i, j) = 1 and MT(j, k) = 1 leads to MT(i, k) = 1. It is antisymmetric: MT(i, j) = – MT(j, i). It has only one line containing zero value 1, and only one line containing a value 1, only one line containing two values 1, …, only one line containing (n – 1) values 1. We have the same property for columns and for values – 1. Because of the antisymmetric property, we may only store the top diagonal matrix; however, for a better understanding, the extended matrix is used here.

Generation of initial population

We can use a totally random generation presented in section 4.2.1 and transcribe the sequence obtained into a permutation matrix or vector. However, in order to

obtain good initial individuals for the sum of weighted tardiness criteria, we can also use specific heuristics for this problem. For example increasing due date sequence (EDD algorithm for earliest due date) and in the same way, decreasing penalty sequence, possibly improved with a simple neighborhood improvement procedure (see Chapter 3).

Crossover operators

We can again use the 1X crossover operator which retains sequences well. It particularly confirms property 1 described below.

Property 1. If i precedes j in "father" and "mother" individuals, then i precedes j in children created by the crossover.

To retain relative operation sequences, it is possible to generalize the 1X crossover operator with the k point crossover operator noted as kX in the following way and illustrated in Table 4.8 for $k = 2$. Explanations are given for the conception of son 1 and son 2 from father and mother (to obtain daughter 1 and daughter 2, simply reverse the roles of father and mother). k crossover points are randomly chosen in order to cut off individuals with $k + 1$ sub-chromosomes where we will distinguish odd and even position sub-chromosomes. For son 1 (resp. son 2), values in even position sub-chromosomes (resp. odd) are those found in the father's sub-chromosome in the same position. For son 1 (resp. son 2), odd position sub-chromosome values (resp. even) are the values shown in this father's sub-chromosomes, but sorted in the mother's sequence. This kX crossover confirms property 1 and it retains the partial sequences of the father (for sons) or of the mother (for daughters) all the more as k is large.

	Position 1		Position 2			Position 3		
Father coding	2	4	3	5	1	7	8	6
Mother coding	6	2	5	3	8	1	7	4
Son 1 coding	2	4	3	5	1	6	8	7
Daughter 1 coding	2	6	5	3	8	4	1	7
Son 2 coding	2	4	5	3	1	7	8	6
Daughter 2 coding	6	2	3	5	8	1	7	4

Table 4.8. *kX sequence crossover*

It is also possible to define a uniform type crossover by using the MT permutation. A crossover is said to be uniform when values from the father and/or the mother are used in an irregular and generally random manner and not by complete sub-sequences to constitute the children.

1	2	3	4	5	6		1	2	3	4	5	6
0	1	1	1	1	-1	1	0	1	1	1	1	1
-1	0	-1	-1	1	-1	2	-1	0	-1	-1	-1	-1
-1	1	0	1	1	-1	3	-1	1	0	1	1	1
-1	1	-1	0	1	-1	4	-1	1	-1	0	1	-1
-1	-1	-1	-1	0	-1	5	-1	1	-1	-1	0	-1
1	1	1	1	1	0	6	-1	1	-1	1	1	0

Table 4.9. *MT "father" and "mother" matrix*

1	2	3	4	5	6		1	2	3	4	5	6
0	1	1	1	1	0	1	0	1	1	1	1	-1
-1	0	-1	-1	0	-1	2	-1	0	-1	-1	-1	-1
-1	1	0	1	1	0	3	-1	1	0	1	1	-1
-1	1	-1	0	1	-1	4	-1	1	-1	0	1	-1
-1	0	-1	-1	0	-1	5	-1	1	-1	-1	0	-1
0	1	0	1	1	0	6	1	1	1	1	1	0

Table 4.10. *SIM and MT "son" matrix*

Tables 4.9 and 4.10 illustrate the crossover called MT2. Permutation values for the father and the mother are respectively: 613425 (from Figure 4.2) and 136452.

Crossover begins by adding father and mother matrices and dividing this sum by 2. When the father and mother have the same value, we retrieve this value, otherwise we obtain zeros (see left matrix in Table 4.10, called the SIM matrix), zeros that will need to be replaced by 1 or – 1 in order to obtain a permutation matrix once again. We then complete the matrix with the following iterative algorithm:

while there is a zero element outside of the diagonal **do**
Randomly choose one of these elements.
For this element, choose the value of the father (with probability π) or the value of the mother (with probability 1-π).
Put opposite value in the symmetric element.
Make matrix transitive.
End while

In this example, we randomly choose for example element (1, 6), then with probability $\pi = 0.6$, we randomly choose value – 1 from the father, i.e. 6 is placed before 1. By transitivity, 6 is also before 3. For the last zero element (2, 5), we presume that the random choice has selected value – 1 from the mother, i.e. 5 is placed before 2. All values are chosen and the permutation obtained is 613452.

From parents 613425 and 136452, we have obtained child 613452. The number of times we have retained the sequence (i, j) in the son as it existed in the father and mother is 9 (number 1 in the left hand matrix) and the number of times we have retained the sequence (i, j) in the son when it existed in the father and/or the mother is equal to the maximum value of 15 ($n(n-1)/2$). This crossover attempts to retain sequences.

Mutation operators

We can use the same operators as in section 4.2.1. Since the goal of mutations is to create diversity, it is not necessary to develop sophisticated approaches in order to be efficient. On the other hand, mutations often ensure connectivity of the space visited, i.e. we can visit the solution space by only using permutations as a neighborhood technique. This is interesting because in general, it is very difficult to demonstrate that crossover operators alone ensure this connectivity.

4.2.3. *Example 3: sum of weighted tardiness and setup times*

Problem description

A one-machine scheduling problem is considered here which is a generalization of example 1 (presence of setup times depending on the sequence of operations) and example 2 (the criteria used is sum of weighted tardiness). We add setup times in the Gantt chart of Figure 4.2.

Preliminary analysis

A solution is considered good because setup times used are not too large and/or because short due date and/or high penalty operations are placed sufficiently early in

the sequence. If we do not want the efficiency of the genetic algorithm to be left to chance, crossover operators must be developed to find a compromise between retaining arcs (corresponding to short setup times) and retaining partial sequences (corresponding to sequencing the most urgent operations first if they lead to high tardiness penalties). In this case, it is possible to use the content of the chromosome describing a solution as well as the problem statement providing setup times, processing times, due dates and penalties with crossovers. The crossover then becomes "data-dependent". Without going into a full discussion on this type of crossover, here is a general idea: when there is a possibility of choosing between retaining part of the father code or part of the mother code, in order to make this choice, we use unbalanced probabilities obtained from supplementary evaluations which use data from the problem and consider partial consequences on the criteria value of each choice.

Possible solution codings

Since it is always permutation scheduling, we can use the permutation vector as well as the permutation matrix.

Generation of initial population

We use identical processes as in sections 4.2.1 and 4.2.2: a totally random generation of permutations, or heuristics which attempt to empirically build good quality solutions. Solutions which can be improved with a few iterations from a local neighborhood technique (see Chapter 3).

Crossover operators

Here we propose a new operator called MT2mod. It is a mix of 2X and MT2. We describe it for the son (simply change father for mother to obtain the daughter). Once we have chosen two crossover points, we copy the central part of the father as the central part of the son. Any operation preceding this central part in the father will remain in front of this central part in the son and any operation which follows this central part in the father will follow this central part in the son. All these precedences are used to complete the matrix to the left of MT2mod crossover (we retain the common partial sequences in the father and in the mother, in addition to the central part of the father and the relative position of father operations in relation to the copied central part). We then finish the MT2mod crossover with the regular procedure by increasing the probability of using the value in the mother as in section 4.2.2. To obtain a data-dependent crossover, we can try to choose the one for which the sum of setup times divided by the number of sequence operations is minimal as central part in the father.

Mutation operators

We can use the same operators as in section 4.2.1.

4.3. Job shop problems

Problem description

This is a shop containing several machines, and each machine has different functions. In order to carry out job *i*, it is necessary to execute a set of n_i operations, where each one requires a specific machine in the shop. The j^{th} operation in job *i* requires p_{ij} units of time on machine M_{ij}. Several criteria can be considered to assess the performance of job shop scheduling; we can use the same as previously described and more specifically the sum of weighted tardiness compared to due dates.

Preliminary analysis

Table 4.11 provides the description of a job shop problem where a solution is provided by the Gantt chart in Figure 4.3.

i	p_{i1}	M_{i1}	p_{i2}	M_{i2}	p_{i3}	M_{i3}	p_{i4}	M_{i4}	d_i	w_i
1	2	M_1	3	M_2	1	M_1	3	M_3	10	2
2	1	M_3	1	M_1	3	M_2	/	/	8	1
3	2	M_2	4	M_1	3	M_3	/	/	10	3

Table 4.11. *Job shop scheduling problem statement*

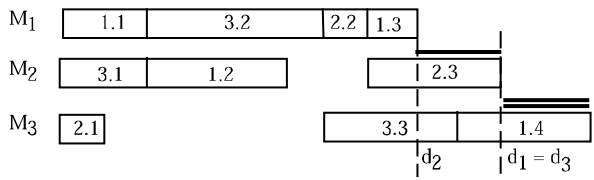


Figure 4.3. *Gantt chart of the job-shop scheduling problem with sum of weighted tardiness*

A solution is considered good because it places the most urgent and highest tardiness penalty jobs early enough. We presume that we are working on a dominant set. For example, we can consider the set for semi-active schedules (by definition, any operation is left shifted, either after the operation preceding it in its job, or after

the operation preceding it on the machine used), thus, for each solution described by a Gantt chart, there corresponds a list of permutations, as each permutation sequences operations for each machine. However, the reverse is not true: if we have a sequence of operations on machines, this still does not define a schedule. For example, no feasible schedule for the description of Table 4.11 corresponds to the list of permutations in Table 4.12. As, according to the jobs, 1.1 precedes 1.2 and 3.1 precedes 3.2 and according to the sequence on machines given by permutations, 3.2 precedes 1.1 and 1.2 precedes 3.1. We therefore have a process as shown in Figure 4.4 and no scheduling is possible because for example, 1.1 only precedes itself.

In a job shop scheduling problem, the choice for the sequence of operations on machines must not create a circuit in the precedence graph built from sequences of operations in jobs and sequences of operations on machines.

M_1	3.2	1.1	2.2	1.3
M_2	1.2	3.1	2.3	/
M_3	2.1	3.3	1.4	/

Table 4.12. *Scheduling resulting from a list of permutations*

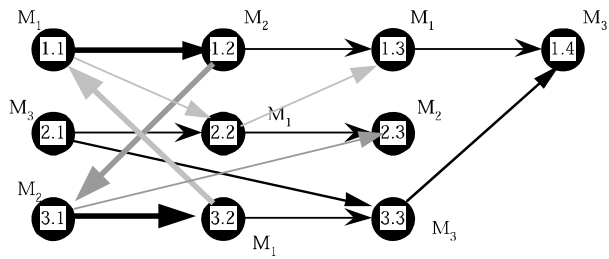


Figure 4.4. *Appearance of a circuit in the precedence graph resulting from jobs and permutations on each machine*

Possible solution codings

There are several solutions to face this issue. The first one (not the best) consists of working on codings of lists of operation permutations on machines and deleting all individuals who do not correspond to feasible solutions because of the presence of circuits in the precedence graph. The second one consists of repairing these individuals with generally empirical procedures which make circuits disappear. The third one,

generally used, consists of considering that permutations on machines are not imperative sequences between operations, but only priorities between operations, priorities used to build a feasible schedule from the chromosome by using a solutions generator. The most frequently used generator consists of developing the schedule with no delay corresponding to these priorities. Its operation is simple: we start at moment 0 with an empty schedule. We then increase the time and we search for the first moment where a free machine has operations waiting in front of it. We then allocate the highest priority operation (the one placed the earliest in the permutation). The problem with proceeding that way is that we only scan on no delay schedules and this set is not dominant for the criteria considered here (the optimal solution may not belong to this set and in this case, the genetic algorithm may not be able to find it). A generator of "active" schedules that we do not present here (see Chapter 2), explores a dominant set of solutions. Other solutions can be used making it possible to avoid creating a circuit while retaining coding which scans a dominant set of solutions (i.e. containing at least one optimal solution). It is the generalized permutation coding [BIE 95] (not presented here) and the ternary coding (0, 1, –1) already proposed in section 4.2.2 for the particular case of permutations and for which it is possible to develop crossover operators that never create circuits in the precedence graph.

Generation of initial population

If coding is assumed to contain priority lists which are transformed into solutions with the help of a generator, then this is called indirect coding. In this case, the coding for the genome (chromosomes) and that for phenotype (corresponding solutions) are no longer in bijection. In order to generate random values for genomes, we must generate random permutations for operations on each machine and complete this set by using the sequence of operations on each machine obtained by applying one or more heuristics.

If we use a ternary MT matrix for coding then $MT(i, j) = 0$ means that there is no precedence relation between operations i and j, directly nor by transitivity. $MT(i, j) = 1$ means that operation i precedes operation j in the precedence graph, either directly because i precedes j in the job or on a machine, or indirectly (for example, i precedes k on a machine, k precedes l in its job, l precedes j on another machine); $MT(i, j) = -1$ if $MT(j, i) = 1$.

It is possible to code any solution obtained by a heuristic in the form of an MT matrix. It is also possible to generate random feasible solutions by working directly on matrix MT. We initialize matrix MT to 0. We then introduce in the MT matrix relations of precedence corresponding to jobs (this sub-graph must be transitive). We then consider MT sub-matrices consecutively corresponding to operations executed on one machine. These sub-matrices must be permutation matrices since we must decide the exact sequence on each machine. We search for a 0 outside of the diagonal in any

of these sub-matrices. We arbitrarily decide to give this element the value 1 or value –1, then we execute a transitive closing on matrix MT. We iterate this process as long as there is a 0 outside of the diagonal in one of these sub-matrices. For the example in Table 4.11, Table 4.13 presents the random generation of an initial solution characterized by the sequence 22, 11, 32, 13 on machine 1, sequence 31, 12, 23 on machine 2 and sequence 21, 33, 14 on machine 3.

		M_1				M_2			M_3		
		11	13	22	32	12	23	31	14	21	33
M_1	11	0	1			1			1		
	13	-1	0			-1			1		
	22			0			1			-1	
	32				0			-1			1
M_2	12	-1	1			0			1		
	23			-1			0			-1	
	31				1			0			1
M_3	14	-1	-1			-1			0		
	21			1			1			0	
	33				-1			-1			0

a) Part of the MT matrix imposed by jobs

	11	13	22	32	12	23	31	14	21	33
11	0	1	-1	1	1	1	0	1	-1	1
13	-1	0	-1	-1	-1	0	-1	1	-1	0
22	1	1	0	1	1	1	0	1	-1	1
32	-1	1	-1	0	0	0	-1	1	-1	1
12	-1	1	-1	0	0	1	-1	1	-1	0
23	-1	0	-1	0	-1	0	-1	0	-1	0
31	0	1	0	1	1	1	0	1	0	1
14	-1	-1	-1	-1	-1	0	-1	0	-1	-1
21	1	1	1	1	1	1	0	1	0	1
33	-1	0	-1	-1	0	0	-1	1	-1	0

b) Randomly generated part
(consecutive random choices only in gray parts:
(11, 22) = –1, (11, 32) = 1, (13, 32) = –1, (12, 23) = 1, (12, 31) = –1, (14, 33) = –1)

Table 4.13. *Generation of a random ternary matrix*

Crossover operators

For indirect coding made up of permutation lists used as priority lists with one solution generator, permutation crossovers presented for one-machine problems can be executed on the coding of each machine (for example, a 1X crossover). For direct ternary coding, we use the MT3 crossover which is a generalization of the MT2 crossover. This crossover starts by adding up the ternary matrices of the father and mother. We then divide the matrix obtained by 2 by rounding to 0 each time we obtain +1/2 or –1/2 (we then retain all compatible sequences for the father and mother). In the resulting matrix, we consider sub-matrices corresponding to each machine (see gray parts in Table 4.13b). For each 0 outside of the diagonal in these sub-matrices, we use the value located in the father or mother with probability 1/2. Each time we add a new value in the matrix, we also add necessary values so that the matrix remains transitive. We iterate the previous process until there is no longer a 0 outside of the diagonal in the sub-matrices corresponding to machines.

Mutation operators

For indirect coding, we use traditional mutation operators for permutations. For direct coding with the help of a ternary matrix, the mutation operator works in the following way. We begin by filling the MT matrix with the mutant in creation with the precedence constraints issued from jobs (as in the case of the generation of an initial solution). We then randomly choose an element off diagonal in one of the sub-matrices corresponding to one of the machines. For this element, we use the opposite value of the MT matrix corresponding to the individual to mutate (+1 if –1 and vice versa). We then continue as we would for a (single parent) crossover. For each 0 outside of the diagonal in a sub-matrix of the mutant, we use the value located in the MT matrix of the individual to mutate. Each time we add a new value, we add the necessary values so that the matrix remains transitive. We reiterate the previous process until there is no longer a 0 outside of the diagonal in the sub-matrices corresponding to machines.

4.4. Hybrid flow shop

4.4.1. *Specific case: one-stage total duration problem*

Description of the problem and preliminary analysis

The one-stage hybrid flow shop problem (see a detailed presentation in Chapter 9) is no more than a parallel machine problem. In this case, each job consists of only one operation. This is the context where all machines are identical or, in other words, where the processing time of a job (or operation) is the same on all machines. The criterion to minimize is duration time C_{max} corresponding to the scheduling of a

set T of jobs. We are in the presence of a pure allocation problem and the sequence of jobs on machines is of no interest here.

When we analyze the nature of a problem solution, we may very easily establish that there is at least one optimal solution (minimizing total scheduling time or makespan) for which any machine executes at least one operation (we presume that there are obviously more operations to be executed than there are machines). This is only true when the machines are all identical and always available. We will only build solutions where no machine will be left inactive (dominant set). In addition, the total load will have to be distributed on all machines without necessarily authorizing an operation to be executed simultaneously on several machines (no splitting) or to be preempted by other operations (no preemption).

Possible solution codings

The problem to solve is to find for each job T, the machine that will execute it in order to minimize total scheduling time. There are several ways to code a solution for this type of problem. We present three of them.

One of the possible approaches to code such a solution is to use a vector where each element represents a job and shows the number of the machine on which it will be executed. This coding will be called DC1-ASS (for Direct Coding for ASSignment) from now on. To give an example, say we have a set of jobs defined by $T = \{1, 2, 3, 4, 5\}$. Suppose we have 3 parallel machines and those processing times are $p_1 = 3$, $p_2 = 5$, $p_3 = 8$, $p_4 = 2$, $p_5 = 4$. A possible solution for this problem is represented by vector S_1 = (1, 2, 3, 2, 1). It corresponds to assigning jobs 1 and 5 to machine 1, jobs 2 and 4 to machine 2 and job 3 to machine 3. Figure 4.5 illustrates coding for this solution as well as the associated Gantt chart. This coding is a direct coding because it describes a solution directly and completely.

We can also build another type of direct coding that is more interesting because it enables us to delete, in the case of identical parallel machines, the randomness linked to machine numbering. This coding, called DC2-ASS, consists of building a matrix in which element (i, j) of the matrix has a value of 1 if jobs i and j are on the same machine and 0 otherwise. This type of coding is even more efficient when there are additional job constraints. It is to be noted that saying two jobs are on the same machine does not lead to any decision in terms of the sequence of these two jobs.

Another "indirect" type of coding, called IC3-ASS for Indirect Coding for ASSignment, consists of establishing a list of priorities for all jobs provided by a permutation vector, for example. This solution requires the use of a solution

generator. Its objective is to determine a solution from a given coding. In the very specific case involved here, the no delay solution generator visits a dominant set of solutions. It is so simple that is seems natural to use it.

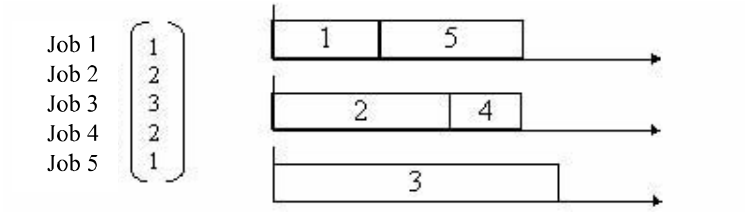


Figure 4.5. *Coding of a solution for one-stage hybrid flow shop and associated Gantt chart*

Generation of initial population

To generate an initial population in the case of direct coding, we must find a set of solutions where each one represents an allocation of all jobs to machines. In order to do this, we can use a method that randomly generates many individuals and which respects the dominant condition "no inactive machine". For each solution, we must calculate the value of criteria C_{max}. This evaluation is very simple because we only need to use the last end date obtained on one of the machines. For indirect coding, we can use the same initial population generation as in section 4.2.1 by applying it to all jobs of *T*.

Possible crossovers

For IC3-ASS coding, we can use crossovers from sections 4.2.1 and 4.2.2. For this type of problem and for DC1-ASS coding, it is not necessary to use a complicated crossover operator. For example, the simple one-point crossover is sufficient. Of course this operator will not guarantee that we obtain two valid children in terms of what has been presented in the preliminary analysis. In fact, it is possible that this type of crossover generates a solution in which a machine may be idle. In this case, we end with a "repair" phase which moves a job to any idle machine.

We use the same data as in the previous example. Figure 4.6 represents a one-point crossover of two individuals. For child 2, no machine is sitting idle and consequently, it respects the dominant condition. It is therefore not necessary to apply a corrective operator.

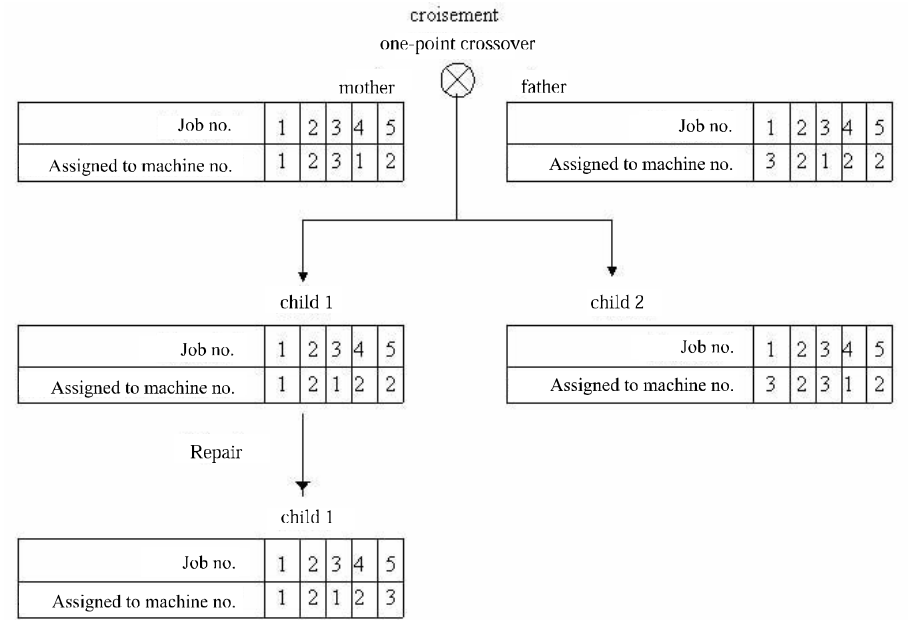


Figure 4.6. *Crossover operator for one-stage hybrid flow shop*

On the other hand, child 1 will have to have a correction to assign a job to machine 3. This repair phase works in a similar way as a mutation operator. In fact, its objective is to identify idle machines. Subsequently, we choose the machine with the greatest load among those with more than one allocated job. We then reallocate a job to an idle machine. This process is repeated until all machines are busy with at least one job. We can then observe that the corrective operator must only be applied on child 1; machine 3 is the only idle one. For this operator, we can choose the longest job (because it cannot be interrupted and will have to be executed by the same machine) among jobs allocated to the machine with the largest load. In our example, the machine with the largest load is machine 2 and the longest job is job 3. Therefore child 1 becomes tuple (1, 2, 1, 2, 3). This correction operator is applied after any crossover, but also after the use of the mutation operator. In the case of DC2-ASS coding, it is preferable to use the MT3 crossover operator, which was presented in section 4.3 and to make it symmetric (–1s become +1s). The repair consists of splitting a subset of operations executing on a single machine in two, until all machines are used.

Mutation

Concerning the mutation operator, clearly, coding a simple probability of modifying a gene in the chromosomes is sufficient, followed by the corrective

operator if necessary for DC1-ASS. The value of this probability must be relatively low so as not to significantly modify the nature of individuals. In the case of DC2-ASS coding, it is more complex because we must retain transitivity of the "is on the same machine as" relation. For IC3-ASS coding, we use techniques from section 4.2.1.

4.4.2. *General case: k stages total duration problem*

Problem description

The K stages hybrid flow shop is a flow shop – all jobs need a set of operations and operation j of each job requires machine j – moreover each operation j must be performed on a single machine chosen from a set of parallel machines dedicated to this operation and grouped on the stage j. This problem comes down to simultaneously solving an allocation problem, an example of which was presented in the previous section, and a sequencing problem. In fact, for each stage, we must define all jobs to be allocated to each machine and for each machine, in which sequence jobs allocated to this machine must be allocated. We focus here on stages made up of identical parallel machines and the criterion to optimize is the total scheduling time.

Preliminary analysis

Before proposing coding for a solution for this type of scheduling, we must first define dominant conditions in order to use coding appropriate for its characteristics. As before, we must never leave a machine idle, so that all available machines are used. Subsequently, as far as priorities to determine sequences on machines are concerned, they must not be too different between two consecutive stages. In fact, the consequence would be the generation of unnecessary inactive times. This is particularly possible using relative job positions. In this way, we can hope that if job i precedes job j at stage h, then either i precedes j at stage $h + 1$, or i and j are on different machines. This analysis will make it possible to offer a crossover operator that is better adapted for this type of problem, and will best retain the sequences.

Figure 4.7 illustrates a case where sequences between two stages are retained. Figure 4.8 shows the same problem for which we have deliberately reversed two sequences between the two stages. In this case, the value of the end scheduling date is more important and does not correspond to an interesting scheduling. We see the inactivity generated by these different sequences. However, this situation is still not verified, it all depends on the distribution of job processing times.

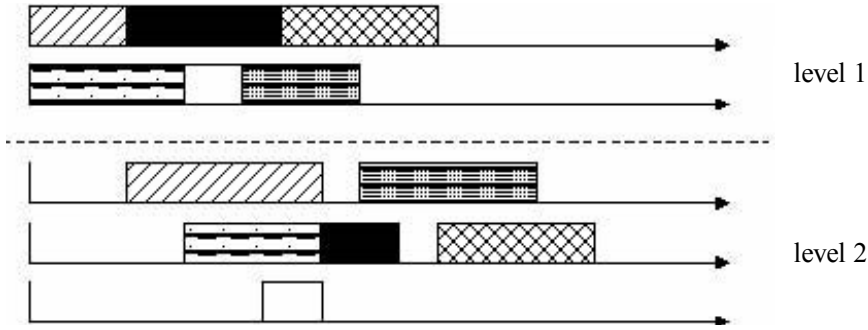


Figure 4.7. *Case where sequences are retained between the two stages*

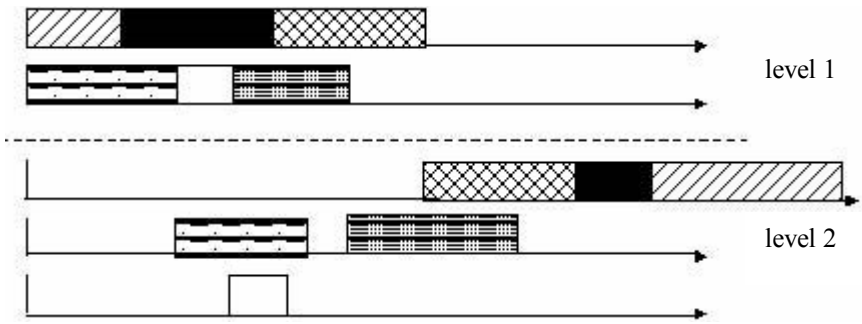


Figure 4.8. *Case where sequences are not retained between the two stages*

Possible solution codings

In this section, we propose two types of coding. In both cases, we consider two chromosomes, one for each family of unknowns for the problem to solve (assignment and sequencing).

The first coding builds a genotype made up of two chromosomes: the first one models the assignment problem by using DC1-ASS coding presented for the one-stage hybrid flow shop problem. The second chromosome determines relative sequences of all jobs for each stage. We actually need an MT3 type matrix (see section 4.3) for each stage. Suppose we focus on stage h. Each element (i, j) of this matrix is equal to either – 1 in the case where operation of job j precedes the one from job i, or to 0 in the case where we do not know the relative position between operations of jobs i and j, or even to 1 in the case where the operation of job i precedes the one from job j. An example of this matrix and its associated Gantt chart is illustrated in Figure 4.9.

We should point out that at each modification made on this matrix, a transitive closing calculation must be applied in order to verify the absence of a circuit (a modification may add a circuit, i.e. job *i* precedes job *j* and job *j* precedes job *i* for a given sequence).

	1	2	3	4	5
1	0	1	0	1	0
2	-1	0	0	-1	0
3	0	0	0	0	-1
4	-1	1	0	0	0
5	0	0	1	0	0

Job 1	1
Job 2	1
Job 3	2
Job 4	1
Job 5	2

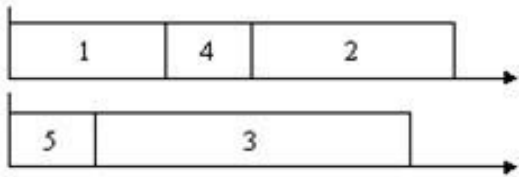


Figure 4.9. *Example of direct coding for hybrid flow-shop and stage 1*

Figure 4.10 presents a solution for the two-stage hybrid flow shop problem for which we have two machines in the first stage and one in the second. This corresponds to a possible solution but not to the best solution.

Another way to code a solution for this problem is to use a priorities-based coding. In this case, we arrive at an indirect coding, implying that it is necessary to implement a solution generator which will successively allocate jobs to machines by using all priorities.

For the hybrid flow shop problem, we can use two chromosomes as before. The first one will give the machine number where each job in each stage will be assigned. The second one will give the priority on this machine for each job and each stage. An example of this coding is given in Figure 4.11.

For stage 1

	1	2	3	4	5
1	0	0	0	-1	0
2	0	0	1	0	1
3	0	-1	0	0	-1
4	1	0	0	0	0
5	0	-1	1	0	0

Job 1	2
Job 2	1
Job 3	1
Job 4	2
Job 5	1

For stage 2

	1	2	3	4	5
1	0	-1	-1	-1	-1
2	1	0	1	1	1
3	1	-1	0	-1	-1
4	1	-1	1	0	-1
5	1	-1	1	1	0

Job 1	1
Job 2	1
Job 3	1
Job 4	1
Job 5	1

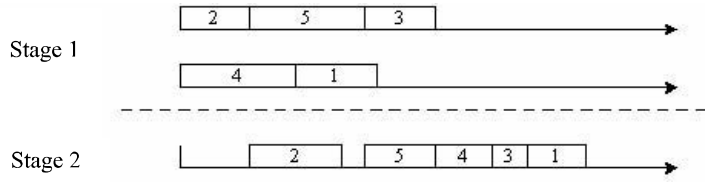


Figure 4.10. *Example of a solution for a two-stage hybrid flow shop*

As in the case with the job shop problem, the problem lies in the choice of a good generator. We can particularly use an active scheduling generator [BAK 74] or a no delay scheduling generator (which never leaves a machine unoccupied when an operation is available to be executed on this machine). These generators respect the choice of machines defined by the first chromosome from the genome, jobs (an operation can only be executed after the end of the previous one) and priorities provided by the second chromosome (when two operations are in conflict for the generator choice, it is always the one with the highest priority that is chosen). With a hybrid flow shop, instead of handling the different stages in parallel as time progresses, we can decompose the resolution by first placing the operations at stage 1, then at stage 2, etc. This will not change the result, but makes it possible to have a better understanding of the advantages and disadvantages of the different generators.

Jobs	1	2	…	N	…	1	2	…	N
Machines	$M_1^{(1)}$	$M_2^{(1)}$	…	$M_N^{(1)}$	…	$M_1^{(k)}$	$M_2^{(k)}$	…	$M_N^{(k)}$
Priorities	$P_1^{(1)}$	$P_2^{(1)}$	…	$P_N^{(1)}$	…	$P_1^{(k)}$	$P_2^{(k)}$	…	$P_N^{(k)}$
	Stage1				….	Stage k			

Figure 4.11. *A second coding for hybrid flow shop*

In the case of a hybrid flow shop, choosing sequences of operations on the different machines at each stage never creates a circuit in the precedence graph as was the case with the general job shop problem (Figure 4.4). We can use priorities associated with operations assigned to each machine in each stage in order to define sequences on this machine and strictly respect the sequences obtained. We then shift to the left each operation in stage h either on the operation preceding it at stage $h-1$, or on the operation preceding it on the machine. We obtain a scheduling called semi-active. In this case, we consider the coding to be direct, since priorities contained in the second chromosome directly give the sequence on machines by simply considering them in increasing order for each machine. It may be interesting to use this type of coding for exploring a dominant solution space; however, many solutions of this type are of very low quality, because to take any sequence on each machine in each stage can generate important idle times on machines, as illustrated in Figure 4.8, for example. That is why genetic algorithms using no delay or active scheduling generators generally converge faster toward good solutions [VIG 96]. However, direct coding can, given sufficient time, provide a better solution (especially when we compare with algorithms using no delay schedules, which a great many authors use, but which are not dominant and can therefore not contain the optimal solution).

Generation of initial population

The generation of initial population can be done by using one or the other coding. Part of the population can be generated in a completely random manner and the other part by using good heuristics.

Possible crossovers

Crossover operators will obviously depend on the chosen coding. In the case where the first coding would be chosen, an operator for the assignment chromosome and/or an operator for the sequencing part are needed. In this way, either the assignment chromosome changes, or the sequencing chromosome, or both. For the assignment chromosome, a one-point crossover is sufficient. For the chromosome

providing sequences, we can use the MT3 crossover operator discussed previously (see section 4.3). In this case however, we must use a repair operator. In fact, it is possible that the use of these two operators can leave a machine idle, which is contrary to the rule of dominance highlighted by the hybrid flow shop problem. Using the repair operator makes it possible to have a solution which respects constraints of dominance. We should point out that it is always interesting to apply the transitive closing calculation as soon as a modification is made on the sequence chromosome.

When the second coding is used, there can be three possible crossover operators: we either apply the one-point crossover in the priority chromosome without changing assignments; or we apply the one-point crossover in the assignment chromosome without changing sequences, but possibly finishing with a repair. And finally, we can apply the one-point crossover anywhere on the genome. In the last case, the assignment and priority chromosomes are modified. If assignments are decided by the generator, the coding and the crossover operator only apply to priorities. It is then not always necessary to develop a repair operator; we must examine in detail if the generator will be able to satisfy conditions of dominance from a genome. If that is not the case, we can possibly use the repair operator previously seen.

Mutation

Concerning mutation, the two codings must be differentiated as with crossover operators. If we work with the first coding, we can work only on the scheduling chromosome. We then randomly choose one machine from all of them. Then we modify with a probability of mutation the sequence assigned to this machine. We can also consider the modification of the chromosome dedicated to assignment; in this case, we then need to use the repair operator seen by the crossover operator. For the second coding, we simply need to choose a low probability to mutate a gene in each chromosome.

Another method that is interesting for this problem is to use a mutation where the probability varies periodically between 0 and 1 [VIG 96]. In this way, the mutation rate is assimilated with a decreasing and periodic function enabling a periodic reinitialization of the population. This can help in getting out of a local extremum. Of course, this principle can be used for other problems and is not specific to the hybrid flow shop.

4.5. Hybrid genetic algorithms

4.5.1. *Hybridization with other metaheuristics*

A large number of crossovers were proposed between genetic algorithms and other metaheuristics [DOM 98, GAL 99, LAS 93, MÜH 91, SUC 87], in particular with neighborhood improvement methods such as the largest gradient, the tabu method or simulated annealing (see Chapter 3). A steep ascent or stochastic descent method is interesting because it makes it possible to verify if better solutions can be found in a neighborhood close to a solution belonging to the population, with low calculation cost, and if that is the case, to possibly improve the value of the best solution found. The question thus raised is this: must the best solution obtained by neighborhood improvement replace the individual thus improved in the population or not? Certain authors systematically apply a neighborhood improvement method and systematically replace original individuals with improved individuals at each genetic algorithm generation. In this case, we can make the following observations:

– the population consists of local *optima* in relation to the considered neighborhood; there is therefore quick but premature convergence to a limited set of local *optima*;

– as the population is mainly improved by the application of the neighborhood method, mutations become improvers and crossover operators become diversifiers: trying to recreate diversity in the population to extract from local *optima* into which the neighborhood method leads them frequently.

Other authors systematically apply a neighborhood improvement method to all individuals in the population at each genetic algorithm generation. The best solution is carefully put aside even if the corresponding individual is not retained. In fact, in this case, the improved individual only replaces the original individual with a generally low probability: 1/20 or 1/10 appear to be good choices according to some authors. At each generation, we can also improve a population's individuals based on a given probability or, which amounts to the same thing, only use the neighborhood method every k generations. This limits algorithm cost. These last hybridizations between genetic algorithms and neighborhood improvement methods converge more slowly toward very good solutions, but are not as quickly trapped by local *optima*. In addition, they respect the spirit of genetic algorithms better since the crossovers must try to preserve the parents' good properties and thus develop quality individuals, and mutation's main role is to create diversity in the population.

4.5.2. *Hybridization with combinatorial optimization methods*

One of the major drawbacks with genetic algorithms is that we cannot demonstrate the optimality of the resulting best solution. In addition, genetic algorithms visit a set of solutions. The size of these sets is generally very large and consequently crossing this type of method with methods based on reducing solution spaces, by pairing operations research and genetic algorithms is interesting.

A crossover used by several authors [DJE 96, POR 98] consists of using a genetic algorithm in a branch and bound procedure (BBP). The objective is to improve the upper boundary (smallest solution found in the case of minimization) by exploring the space of solutions corresponding to certain nodes in the arborescence created by consecutive branchings. The idea is obviously not to execute a genetic algorithm at each node in the arborescence, since the computational time for a BBP is high enough. On the other hand, applying it to levels in the research tree in order to improve boundary calculation is interesting. In this way, by respecting the constraints defined by a certain node in the research tree, the goal of the genetic algorithm is to try to improve the value of the upper boundary. Certain branches of the research tree can then be cut, limiting the size of the arborescence explored. The problem with this approach is, on one hand to find appropriate coding to transcribe constraints of a node in the research tree in codings used, and on the other hand, to find a good compromise between the BBP computation time and the time allowed for the improvement of the upper boundaries [DJE 96, POR 98].

4.6. Conclusion

In this chapter, scheduling problems have been considered in an increasingly difficult order in relation to their approximate resolution by genetic algorithms. We have observed, on the one hand, schedules for which all operations are pre-assigned to resources (one-machine and job shop problems) and, on the other hand, schedules for which it is necessary to allocate resources (identical parallel machine and hybrid flow shop problems).

We have also observed problems for which permutation coding can actually give the order of operation on the different resources (immediate direct coding possible for all problems presented except for the job shop problem) problems for which traditional permutation coding no longer makes it possible to describe solutions quickly for which we must either use a generator to transform permutation coding into solutions (indirect coding), or use more sophisticated direct coding such as for example the MT3 matrix. We also attempted to show how we could try to develop genetic operators to make resulting genetic algorithms as efficient as possible, based

in coding and criteria considered. Experiments made in late 1999/early 2000 [POR 00] confirmed that crossovers retaining the good properties of parents, as they were defined in this chapter for the problem described in section 4.2.3, on average create better children than others. The improvement is even more significant when we use "data-dependent" crossovers as for example in [MAH 00] for hybrid flow shop problems with a single machine per stage.

In conclusion, genetic algorithms are robust methods (low sensitivity to variations in digital data from statements), easy to develop and to implement. However, as with neighborhood methods, it is very important to analyze the characteristics of the problem to solve and the characteristics of the solutions offering good values to performance criteria, in order to develop the most efficient coding and genetic operators. Before applying genetic algorithms to concrete problems, experimentation is absolutely necessary. They enable the comparison of several codes and genetic operators in industry or randomly generated examples and the choice of those most suitable based on the problem involved as well as numerical instances with the same characteristics as the concrete application to solve (for example, a very large time range in relation to the setup time range for the problem corresponding to section 4.2.3). The parameters for the general schema retained for the genetic algorithm (size of population, mutation percentage, crossover percentage, etc.) are also retained with this experimentation phase.

4.7. Bibliography

[BAK 74] BAKER K., *Introduction to Sequencing and Scheduling*, John Wiley, 1974.

[BIE 95] BIERWIRTH C., "A generalized permutation approach to job shop scheduling with genetic algorithms", *OR Spektrum*, 17, p. 87-92, 1995.

[DAV 85] DAVIS L., "Job-shop scheduling with genetic algorithms", *1st Int. Conf. on Genetic Algorithms and their Applications*, Lawrence Erlbaum (ed.), Hillsdale, New Jersey, p. 136-140, 1985.

[DAV 91] DAVIS L., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

[DJE 96] DJERID L. and PORTMANN M.-C., "Genetic algorithm operators restricted to precedent constraint sets: genetic algorithm designs with or without branch and bound approach for solving scheduling problems with disjunctive constraints", *Conference IEEE/Systems, Man and Cybernetics (SMC'96)*, October 14-17, Beijing, vol. 4, p. 2922-2927, 1996.

[DOR 98] DORNE R. and HAO J.K., "A new genetic local search algorithm for graph coloring", *Lecture Notes in Computer Science* 1,498, p. 745-754, Springer-Verlag, 1998.

[GAL 99] GALINIER P. and HAO J.K., "Hybrid evolutionary algorithms for graph coloring", *Journal of Combinatorial Optimization*, vol. 3, no. 4, p. 379-397, 1999.

[GOL 89] GOLDBERG D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, Mass., 1989.

[HOL 75] HOLLAND J.H., *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, Mass., 1975.

[LAS 93] LASH S., Genetic algorithms for weighted tardiness scheduling on parallel machines, Technical report 93-01, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, Illinois, 1993.

[MAH 00] MAHDI H., ALPAN G. and PORTMANN M.-C., "Genetic algorithms with data oriented operators to solve flow shop problems having tardiness criteria", *IERC 2000, Industrial Engineering Research Conference*, Ohio, May 21-24, 2000.

[MÜH 91] MÜHLENBEIN H., *Evolution in Time and Space – The Parallel Genetic Algorithm. Foundation of Genetic Algorithms*, Rawlins G.J.E., vol. 1, Morgan Kaufmann (eds.), San Mateo, CA, p. 316-337, 1991.

[POR 96] PORTMANN M.-C., "Genetic algorithms and scheduling: a state of the art and some propositions", *Workshop on Production Planning and Control*, Mons, I-XIV, 1996.

[POR 98] PORTMANN M.-C., VIGNIER A., DARDILHAC D. and DEZALAY D., "Branch and bound crossed with GA to solve hybrid flowshop", *European Journal of Operational Research*, 107, 389-400, 1998.

[POR 00] PORTMANN M.-C. and VIGNIER A., "Performances' study on crossover operators keeping good schemata for some scheduling problems", *GECCO'2000, Genetic and Evolutionary Computation Conference*, Las Vegas, July 8-12, 2000.

[SUH 87] SUH J.Y. and Van GUCHT D., "Incorporating heuristic information into genetic search", *2nd Int. Conf. on Genetic Algorithms*, Grefenstette J.J., Lawrence Erlbaum Associates (eds.), Hillsdale, New Jersey, 100-107, 1987.

[VIG 96] VIGNIER A. and VENTURINI G., "Resolution of hybrid flowshop with a parallel genetic algorithm", *International Workshop on Project Management and Scheduling*, Poznan, p. 258-261, 1996.

[WHI 89] WHITLEY Y., STARKWEATHER T. and FUQUAY D., "Scheduling problems and traveling salesman: the genetic edge recombination operators", *Third Int. Conf. on Genetic Algorithms and their Applications*, Morgan Kaufmann, San Mateo, CA, p. 133-140, 1989.

Chapter 5

Constraint Propagation and Scheduling

5.1. Introduction

5.1.1. *Problem and chapter organization*

By "constraint-based approach" for scheduling problems, we refer to studies focusing on the representation and processing of constraints in scheduling [ERS 76a, FOX 83, DIN 90, LEP 91, SMI 93, FAR 94, LEP 94, NUI 94, VAN 94, CAS 95, BAP 98, BAP 01, BRU 02, DOR 02, LOP 03]. These approaches compare and combine operations research methods (graph theory, mathematical programming, combinatorial optimization methods), with constraint representation and processing from artificial intelligence (constraint satisfaction problems, constraint propagation algorithms, constraint programming languages).

The goal of these studies is to put tools in place to facilitate interaction between models and decision makers, by integrating useful analysis methods in a context of decision support (consistency checking, characterization of solution space) and efficient resolution algorithms (complete generation, optimization).

At the core of concerns common to these studies, we find the problem of developing efficient and general constraint propagation mechanisms, which are the subject of this chapter. Specific results taken from scheduling problems and general results taken from *constraint satisfaction problems* have been grouped together.

Chapter written by Patrick ESQUIROL, Pierre LOPEZ and Marie-José HUGUET.

First we present purely temporal problems. There are complete propagation algorithms with acceptable complexity for *simple* temporal problems, in which each constraint limits the distance between two dates to a connected domain. Processing more evolved temporal constraints, based notably on disjunctive temporal intervals, makes the problem very difficult, even when these disjunctions are limited to two intervals.

Next, we will focus on problems in which tasks must respect temporal and resource sharing constraints simultaneously. We present a summary of *constraint analysis* results [ERS 76a, ESQ 87, LOP 91] which focus on the development of propagation rules making simple temporal constraints and resource sharing constraints interact. Two types of reasoning are presented. The first requires a prior conflict analysis between tasks and deduces sequencing conditions. The second simultaneously integrates constraints of time and resources. It limits location of a task in relation to a time interval in order to respect the energetic balance.

The use and control of constraint propagation mechanisms is discussed. They may be used to discover a global inconsistency in a problem formulation quickly, or to simplify its resolution.

A few extensions are considered at the end of the chapter, the case of preemptive problems and the case of scheduling problems with assignment constraints more specifically.

5.1.2. *Constraint propagation*

The resolution of a combinatorial problem is often reached with a tree search algorithm. At each step, only certain variables receive a value. We will thus talk equally about *assignment* of a value v (to a variable x) or *instantiation* of a variable x (by a value v) to designate a basic decision, and we will denote it by $x \leftarrow v$. Variables that have not yet received a value at any level are called free variables.

To guarantee obtaining a solution – if such a solution exists – it is necessary to list the remaining choices for the variables already instantiated if we want to be able to explore other possibilities in case of a dead end. Control of the search tree development is carried out in a mode using *chronological backtracking* and the resolution is non-deterministic because we cannot predict the choices actually leading to a solution.

The goal of *constraint propagation* is to simplify the resolution of a problem or to demonstrate the absence of a solution. It does not represent a resolution

technique, but it represents a set of constraint rewriting techniques. In practice, this may consist of removing the values not belonging to any solution from the decision variable domain. This domain *filtering* avoids many resolution attempts doomed to fail. These techniques also simplify the expression of constraints, by eliminating *redundant constraints* for example. Finally, propagation can, in certain cases, show proof of *global inconsistency* of the problem before any resolution attempt.

The general schema of a backtracking algorithm is as follows (inspired by [SAD 96] and [NUI 94]):

while *unassigned variables remain* **and** *a global inconsistency is not detected*
    apply propagation rules
    if *inconsistency is detected*
    then if *unexplored choices exist*
        then *backtrack to an unexplored choice*
        else *global inconsistency detected*
    else
        select a free variable
        choose a value for this variable and store the remaining values

Algorithm 5.1. *A general non-deterministic resolution algorithm*

We should emphasize that constraint propagation does not modify all the solution space; it only facilitates its exploration.

DEFINITION.– Two problems are *equivalent* if they have the same sets of variables and the same solutions.

DEFINITION.– A constraint C is redundant in relation to a subset of constraints $E = \{C_1, C_2, \ldots, C_p\}$ if we can show that satisfaction of all E constraints implies satisfaction of C. Between two equivalent problems the only difference is the presence of a redundant constraint.

Constraint propagation transforms a problem into an equivalent problem until its expression stabilizes. Schematically speaking, at each propagation step, the following cycle is repeated:

1) select a subset of constraints and generation of an induced constraint;

2) discover a redundant constraint in relation to the induced constraint;

3) modification of the set of constraints: addition of the induced constraint and deletion of the redundant constraint.

Constraint-based approaches have shown their usefulness in the representation and resolution of NP-hard scheduling problems such as job-shop problems [BEL 89, DIN 90, ERS 76b, ERS 80, SMI 93]. These approaches use general techniques based on constraint satisfaction problems, including constraint propagation, as well as resolution strategies focusing on the choice of variables to instantiate, choice of values for a given variable, backtracking techniques, etc. These last elements greatly condition global resolution algorithm performance.

5.1.3. *Scheduling problem statement*

The idea is to organize the execution of a set of *n* tasks with a set of *m* resources. Each task i is characterized by its duration p_i and must be executed in a time window $[r_i, d_i]$ (r_i is its earliest start time and d_i its latest finish time). Preemption is prohibited: once started, a task cannot be interrupted. On the other hand, each task *i* requires a known and constant a_i quantity (*intensity*) from a single resource k_i (*mono-resource* problem) for its completion. We assume that the allocation problem is solved when: $\forall i$, k_i is known. Resources are renewable (capacity A_k of resource k is known and constant). The constraints considered are temporal constraints (deadlines, precedence between tasks, etc.) as well as resource sharing constraints.

The scheduling problem involved covers disjunctive resource problems (for example, shop floor scheduling) and cumulative resource problems (for example, project scheduling).

5.1.4. *Notations*

The following list of notations is inherent to this chapter; the new concepts referred to will be discussed throughout this chapter. For more general notations, please refer to Chapter 2.

t_i, c_i : start and completion time of a task i

$\underline{t}_i$, $\overline{t}_i$: earliest start time, latest start time of a task i

$\underline{c}_i$, $\overline{c}_i$: earliest completion time, latest completion time of a task i

$\Delta = [t_\Delta, c_\Delta]$: interval of study

T_k : set of tasks using resource k : $T_k = \{i \in T \,/\, k_i = k\}$

n_k : number of tasks using resource k : $n_k = |T_k|$

W_k^{Δ} : maximum production of resource k over interval Δ

w_i^{Δ} : energy required for task i over interval Δ

$\underline{w}_i^{\Delta}$: minimum energy required for (or *compulsory consumption* of) task i over interval Δ

$\overline{w}_i^{\Delta}$: maximum energy required for task i over interval Δ

S_i^{Δ} : available energy for execution of i over interval Δ

5.2. Time constraint propagation

5.2.1. *Introduction*

It is possible to see a scheduling problem as a specific class of *constraint satisfaction problem* (CSP) [NUI 94, VAN 94]. We must determine the assignment of variables representing the start or completion time for each task (both if the task durations are not known in advance) within a domain associated with the time window of the task, in such a way that the set of constraints is satisfied, particularly resource constraints. In artificial intelligence, a particular type of CSP was proposed, *temporal CSPs* or *TCSPs* [DEC 91, DEC 03], for planning and temporal reasoning problems [DAV 87]. In TCSPs, the variables correspond to time intervals or to time events, the constraints connecting two events [VIL 86] or two intervals [ALL 83] correspond to numerical or symbolic relations. Combinations of these two types of relations were proposed in [MEI 96]. Schwalb and Vila review different modelings of temporal problems and associated constraint propagation methods [SCH 98]. In scheduling, temporal constraints traditionally used are *potential inequalities* (see Chapter 2). Afterwards, we only focus on problems where temporal constraints are expressed by numerical relations between time events, including potential inequalities.

5.2.2. *Definition*

A numerical temporal constraint satisfaction problem, also called a *numerical TCSP*, is defined in [DEC 91] by:

– a set of temporal variables $\{x_1, \ldots, x_n\}$ corresponding to instants;

– a set of domains associated with each variable $\{D_1,\ldots,D_n\}$. Each domain D_i represents all values which can be used by x_i; each domain D_i is the union[1] of a set of n_i disjoint intervals: $D_i = I_i^1 \cup I_i^2 \cup \ldots \cup I_i^{n_i}$;

– a set of binary C_{ij} constraints limiting possible values of a distance between two variables, $x_j - x_i$, by domain D_{ij} of n_{ij} disjoint intervals: $D_{ij} = I_{ij}^1 \cup I_{ij}^2 \cup \ldots \cup I_{ij}^{n_{ij}}$. C_{ij} represents constraint $x_j - x_i \in D_{ij}$.

Domain constraints can be put in the form of binary constraints C_{0i} and C_{i0} by introducing a variable x_0 representing the time origin. The universal constraint corresponds to interval $]-\infty,+\infty[$ covering all possible values. An inconsistency appears in the problem when the set of intervals associated with the domain of a variable or constraint is empty.

Some classical shop scheduling problems can be represented by a TSCP in which variables correspond to start times of tasks. Constraints formulated by a conjunction of potential inequalities (due dates and precedence constraints) can be directly translated into binary TCSP constraints associated with a single interval; for example precedence between two tasks i and j corresponds to $t_j - t_i \in [p_{ij},+\infty[$. Resource sharing constraints, which are modeled by a disjunctive set of potentials inequalities disjunctions, for example $(t_j - t_i \geq p_i) \vee (t_i - t_j \geq p_j)$, can be translated into binary TCSP constraints including two intervals: $t_j - t_i \in]-\infty,-p_j] \cup [p_i,+\infty[$. On the other hand, certain resource constraints, such as cumulative constraints, cannot be represented by binary TCSP constraints.

5.2.3. *Simple temporal problems*

From the general definition of TCSPs derives a specific class of problems called *simple temporal problems* (STP) in which all constraints only involve one interval. Thus, in an STP each constraint C_{ij} represents the double $a_{ij} \leq x_j - x_i \leq b_{ij}$ inequality.

DEFINITION.– A problem is *minimal* if all domains associated with variables and constraints do not include any inconsistent value (we can thus say that the domain is minimal).

---

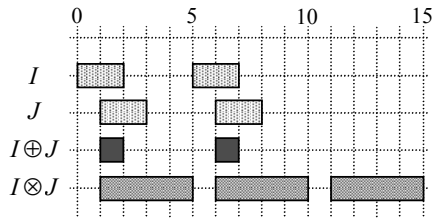
1 By definition, the union (noted as $\cup$) between separate intervals is not an interval (the domain obtained is not connected); we also speak of *domain holes*.

Binary constraints of an STP can be represented in a potentials graph. The minimal problem is thus obtained polynomially by applying a longest path algorithm. In TCSP formalism, the corresponding propagation can be represented from the definition of two operations between constraints (illustrated in Figure 5.1):

– *intersection* operation, denoted $\oplus$, is defined between two binary constraints involving the same couple of variables; let $C_{ij}^1 : x_j - x_i \in D_{ij}^1$ and $C_{ij}^2 : x_j - x_i \in D_{ij}^2$ be the two constraints; the intersection $C_{ij}^3 = C_{ij}^1 \oplus C_{ij}^2$ is defined by constraint $x_j - x_i \in D_{ij}^3$ with $D_{ij}^3 = D_{ij}^1 \cap D_{ij}^2$;

– *composition* operation, denoted $\otimes$, is defined between two binary constraints with one common variable: let $C_{ik} : x_k - x_i \in D_{ik}$ and $C_{kj} : x_j - x_k \in D_{kj}$ be the two constraints; the composition $C_{ij} = C_{ik} \otimes C_{kj}$ is defined by constraint

$x_j - x_i \in D_{ij}$ with $D_{ij} = \{x / \exists x_1 \in D_{ik}, \exists x_2 \in D_{kj}, x = x_1 + x_2\}$.



$I = \{[0,2],[5,7]\}$, $J = \{[1,3],[6,8]\}$, $I \oplus J = \{[1,2],[6,7]\}$, $I \otimes J = \{[1,5],[6,10],[11,15]\}$

Figure 5.1. *Intersection and composition operations*

The following algorithm represents local consistency conditions between any triplet of variables. Applied to an STP, it is complete. It is equivalent to the Floyd-Warshall algorithm applied to the potentials graph corresponding to STP.

Function PC-0(P)
  $P' \leftarrow P$
  for k, i, j **from** 1 **to** n **loop**
    $C'_{ij} \leftarrow C'_{ij} \oplus (C'_{ik} \otimes C'_{kj})$
    if $D'_{ij} = \varnothing$ **then stop** *(inconsistency)*
  return P'

Algorithm 5.2. *PC-0 algorithm for simple temporal problems*

For an STP, this algorithm will help in determining the minimal domain of any C_{ij} constraint. As a comparison, propagation performed by the less complex Bellman-Ford algorithm (see Chapter 2) only determines the minimal variable domains, i.e. constraints of form C_{i0} and C_{0i}.

5.2.4. *General temporal problems*

For general temporal problems, PC-0 is sound (deleted times are really inconsistent), but it is incomplete. Several runs are usually required to arrive at a *fixed point* (problem for which no adjustment of constraint domains is possible). The idea of executing several runs is used in the PC-1 algorithm, which carries out a *path-consistency* propagation [DEC 03] in a numerical TCSP.

Although it improves PC-0 (constraint update is stronger), PC-1 is incomplete and does not guarantee the detection of an inconsistent problem. In addition, the application of PC-1 is limited by the fact that it can lead to a large fragmentation of intervals associated with constraints as illustrated in the following example [SCH 97].

EXAMPLE.– Three binary constraints C_{ij}: $\{[0,22],[23,33],[34,50]\}$, C_{ik}: $\{[1,2],[11,12],[21,22]\}$, C_{kj}: $\{[0,1],[16,17],[23,24]\}$. The application of the PC-1 algorithm to update constraint C_{ij} via variable x_k results in C_{ij}: $\{[1,3],[11,13],[17,19],[21,22],[23,23],[24,26],[27,29],[34,36],[37,39],[44,46]\}$.

Function PC-1(P)
  $P' \leftarrow P$
  inconsistency ← false
  repeat
    End ← true
    for k, i, j **from** 1 **to** n **loop**
      if $C'_{ij} \neq C'_{ij} \oplus (C'_{ik} \otimes C'_{kj})$
      then $C'_{ij} \leftarrow C'_{ij} \oplus (C'_{ik} \otimes C'_{kj})$
        end ← false
      if $D'_{ij} = \varnothing$ **then** inconsistency ← true
  until end **or** inconsistency
  return P'

Algorithm 5.3. *Path-consistency in a TCSP*

In order to avoid this fragmentation problem, Schwalb and Dechter propose two methods [SCH 97]. The first one relies on a relaxation of binary constraints to constraints formed from a single interval covering all possible values. For each constraint $C_{ij} : x_j - x_i \in \{I_{ij}^1, ..., I_{ij}^k, ...I_{ij}^{n_{ij}}\}$, the relaxation proposed is as follows: $C'_{ij} = \text{relax}(C_{ij}) : x_j - x_i \in [\min_{k=1..n_{ij}} (I_{ij}^k), \max_{k=1..n_{ij}} (I_{ij}^k)]$.

This relaxation makes it possible to go back to an STP on which PC-0 determines minimal constraints. The ULT algorithm (upper-lower tightening) is based on this relaxation. For an initial P TCSP, its principle follows the following diagram.

Function ULT(P)
  inconsistency ← false
  repeat
    end ← true
    for i,j **from** 0 **to** n **do** *(relaxation)*
      $C'_{ij} \leftarrow$ relax(C_{ij})
    $P'' \leftarrow$ PC-0(P')
    for i,j **from** 0 **to** n **do** *(P''' = "intersection" between P and P'')*
      $C'''_{ij} \leftarrow C''_{ij} \oplus C_{ij}$
      if $D'''_{ij} = \varnothing$ **then** inconsistency ← true
    if $P \neq P'''$ **then** end ← false
    $P \leftarrow P'''$
  until end **or** inconsistency
  return P

Algorithm 5.4. *Path-consistency without domain fragmentation*

EXAMPLE.– Relaxations of the three binary constraints from the previous example are:

$C_{ij} : \{[0,22],[23,33],[34,50]\} \Rightarrow C_{ij} : \{[0,50]\}$

$C_{ik} : \{[1,2],[11,12],[21,22]\} \Rightarrow C_{ik} : \{[1,22]\}$

$C_{kj} : \{[0,1],[16,17],[23,24]\} \Rightarrow C_{kj} : \{[0,24]\}$

The application of the PC-0 algorithm on constraint C'_{ij} *via* variable x_k results in $C''_{ij} : \{[1,46]\}$. The intersection between C''_{ij} and C_{ij} results in $C'''_{ij} : \{[1,22],[23,33],[34,46]\}$.

A second method for limiting interval fragmentation consists of defining a new intersection operation between two binary constraints, noted as $\triangleleft$. As with $\oplus$, operation $\triangleleft$ is defined between two binary constraints involving the same pair of variables.

Let the two binary constraints be $C_{ij}^1 : x_j - x_i \in I_1^1 \cup I_2^1 \cup \ldots \cup I_{n_1}^1$ and $C_{ij}^2 : x_j - x_i \in I_1^2 \cup I_2^2 \cup \ldots \cup I_{n_2}^2$. Constraint $C_{ij}^3 = C_{ij}^1 \triangleleft C_{ij}^2$ is defined by the set of intervals $x_j - x_i \in I_1^3 \cup I_2^3 \cup \ldots \cup I_{n_3}^3$, $n_3 \leq n_1$, such that $\forall k = 1 \ldots n_3$, $I_k^3 = [lb_k^3, ub_k^3]$, where lb_k^3 and ub_k^3 are the lower and upper bounds of intersection $I_k^3 \oplus \left(I_1^2 \cup I_2^2 \cup \ldots \cup I_{n_2}^2 \right)$. By definition, this intersection operation does not increase the initial number of intervals by constraints but it is not commutative: $\left(C_{ij}^1 \triangleleft C_{ij}^2 \right) \neq \left(C_{ij}^2 \triangleleft C_{ij}^1 \right)$. The LPC algorithm (loose path-consistency), based on this principle, is equivalent to the PC-1 algorithm by replacing $\oplus$ by $\triangleleft$.

EXAMPLE.– From the three previous binary constraints and by applying the LPC algorithm on constraint C_{ij} using x_k: $C_{ij} \leftarrow C_{ij} \triangleleft (C_{ik} \otimes C_{kj})$, the result is: $C_{ij} : \{[1,22], [23,29], [34,46]\}$.

It has been shown in [SCH 97] that the PC-1, ULT and LPC algorithms determine the same lower and upper bounds for each constraint. The difference between these algorithms is in the management of "holes" appearing in intervals associated with the constraints. The PC-1 algorithm causes an increase in the number of holes in the constraint domains. The ULT algorithm retains the actual holes already present in constraints whereas the LPC algorithm enlarges them.

5.3. Resource constraint propagation

Only considering resource sharing constraints defines a *sequencing problem*: tasks requiring the same resources must be partially – sometimes completely – sequenced in order for instant global consumption to remain within resource availability. The consideration of time window constraints reduces the set of sequencing problem solutions; certain configurations are prohibited, others become mandatory, which can lead to tightening of task time windows. Thus, there are propagation rules for resource constraints that we can classify into two methods of reasoning.

The first method requires a prior conflict characterization, or *critical sets of tasks*. Sequencing conditions obtained can be propagated [ESQ 87], and lead to a tightening (or *adjustment*) [CAR 94] of task time windows.

The second is linked to the concept of *energy*, helping to make quantitative methods of reasoning integrating time and resource constraints [ERS 91, LOP 91, LOP 92, LOP 96]. As with the previous method, this second method of reasoning can produce sequencing conditions without requiring an analysis of the problem in terms of critical sets. However, it also makes it possible to prohibit locating a task at certain time intervals – which would generate a deficit in energetic balance. This reasoning becomes interesting when some execution characteristics are not completely known, for example when task durations depend on resources used (see section 5.5.2).

5.3.1. *Characterization of conflicts*

In order to characterize potential conflicts for the use of a resource, we determine critical sets of tasks (called conflict sets in [BEL 82] and forbidden sets in [BAR 88]). These are smaller subsets of tasks which cannot be executed simultaneously because of a lack of resources. Since a critical set is minimal, sequencing two tasks of this set resolves the conflict.

5.3.1.1. *Critical sets*

DEFINITION.– $I \subseteq T_k$ is a *critical set of tasks* for a resource k if and only if: $\sum_{i \in I} a_i > A_k$, and $\forall j \in I$, $\sum_{i \in I \setminus \{j\}} a_i \leq A_k$.

A necessary and sufficient condition for a scheduling to be eligible from a resource constraint point of view is that all critical sets be resolved. The different sequencing decisions taken are compatible with each other; the precedence constraints graph, once updated, must not include any cycle because of anti-symmetry and transitivity of the relation of precedence between tasks. During resolution, consistency checking is performed by temporal propagation mechanisms.

EXAMPLE.– Tasks $\{A, B, C, D\}$ use a resource k available in five copies ($A_k = 5$), according to the following terms:

i	A	B	C	D
a_i	4	3	2	1

The critical sets of this problem are: $\{A,B\}$, $\{A,C\}$, and $\{B,C,D\}$. As an example, resolution of the conflict linked to $\{B,C,D\}$ goes through the choice of one of the following sequencing decisions:

$$(B \prec C) \vee (C \prec B) \vee (B \prec D) \vee (D \prec B) \vee (C \prec D) \vee (D \prec C)$$

5.3.1.2. *Recursive search algorithm of critical sets*

To make it simple, we will only describe critical set search in the case of a single resource[2]. Let E be the list of tasks using the same resource, sequenced by non-increasing intensity (we denote by $a_{E[i]}$ the intensity of the i^{th} task of E): $\forall(i,j) \quad 1 \le i < j \le |E| \quad \Rightarrow \quad a_{E[i]} \ge a_{E[j]}$.

The algorithm develops a binary search tree according to a depth-first search. Each node is characterized by four elements:

– current list of task E which can still be chosen to develop new critical sets;

– accumulation R of intensities of tasks from *E*;

– a set I, grouping tasks already chosen;

– accumulation S of intensities of tasks from *I*.

In a given node, if $R + S \le A_k$, then no critical set containing I can be obtained and this node is not developed. Otherwise, if $S > A_k$, then current set I is a critical set and the node is not developed. Otherwise, we separate the node by creating two successor nodes, one representing all critical sets obtained by completing I by other tasks from E or the first at least, $E[1]$, the other representing all critical sets including I and other tasks from *E* except $E[1]$.

The exhaustive search for all critical sets of a problem presents an exponential complexity. Indeed, the space of critical set search is the one for the parts of T_k sets: the size of this space increases in $O(2^{n_k})$. In practice, even though increasing the problem size does not necessarily lead to an increase in resource capacity, the increase of the number of critical sets remains polynomial [BAR 88].

---

2 In order to find all critical sets, we can apply this algorithm to each of the problem's resources consecutively. This way (resource by resource) may not be the most efficient in a *multi-resource* case (case not considered here, where execution of a task involves several resources simultaneously). In fact, we may discover the same sets several times if their criticality is valid for several resources.

Initialization

$E \leftarrow$ sort(T_k, a_i, non-increasing)

$R \leftarrow \sum_E a_i$; $I \leftarrow \varnothing$; $S \leftarrow 0$

CRIT_SET(A_k, E, R, I, S)

Function CRIT_SET(A_k, E, R, I, S)

if $R + S \leq A_k$ **then return** $\varnothing$

else

if $S > A_k$ **then return** $\{I\}$

else

$I_1 \leftarrow$ CRIT_SET$(A_k, E \setminus \{E[1]\}, R - a_{E[1]}, I \cup \{E[1]\}, S + a_{E[1]})$

$I_{\overline{1}} \leftarrow$ CRIT_SET$(A_k, E \setminus \{E[1]\}, R - a_{E[1]}, I, S)$

return $I_1 \cup I_{\overline{1}}$

Algorithm 5.5. *Search algorithm for critical sets*

In the case of shop scheduling problems, complexity is naturally decreased because all critical sets are disjunctive pairs; for machine k, there remains $n_k(n_k - 1)/2$. In the case of cumulative problems, there can also be critical sets of two tasks. We group them into *maximum disjunctive sets* in which propagation rules for the disjunctive case apply.

5.3.1.3. *Maximum disjunctive sets*

DEFINITION.– We call *maximum disjunctive sets* (MDSs) a maximum subset of tasks D such that each couple of tasks in D is a disjunctive pair.

D tasks must therefore be totally sequenced. MDSs are also called *one-machine problems* [CAR 84]. In fact, from a resource sharing standpoint, everything happens as if each maximum disjunctive set's tasks were running on the same machine. When each task only uses one resource, searching for this type of set is relatively easy. In the case of disjunctive problems, for a resource k, T_k is the only MDS.

In the case of cumulative problems, T_k can include several MDSs. E is the list obtained by sorting T_k by non-increasing intensity. The algorithm used to determine MDSs uses the fact that an MDS contains at most one element from rank j in E such that $a_{E[j]} \leq A_k / 2$.

The sub-list $MDS^0 = E[1,\ldots,p]$ where p is such that $a_{E[p\ 1]} + a_{E[p]} > A_k$ and $a_{E[p]} + a_{E[p+1]} \leq A_k$ corresponds to the first maximum disjunctive set. The others are obtained by considering each task rank $p' \in [p+1,\ldots,n_k]$ and by subtracting from sub-list $E[1,\ldots,p']$ all tasks of rank q so that $a_{E[q]} + a_{E[p']} \leq A_k$.

Function MAX_DISJ_SET(T_k)
  $E \leftarrow$ sort(T_k , a_i , non-increasing); $n_k \leftarrow |E|$; $p \leftarrow 1$
  while $p < n_k$ **and** $a_{E[p]} + a_{E[p+1]} > A_k$
    $p \leftarrow p+1$
  if $p > 1$
  then $MDS^0 \leftarrow E[1,\ldots,p]$ *(the first MDS)*
    $l \leftarrow 1$; $q \leftarrow p-1$; $p \leftarrow p+1$
    while $p \leq n_k$ **and** $q \geq 1$
      while $q \geq 1$ **and** $a_{E[q]} + a_{E[p]} \leq A_k$
        $q \leftarrow q-1$
      if $q \geq 1$
      then $l \leftarrow l+1$
        $MDS^l \leftarrow E[1,\ldots,q] \cup E[p]$ *(the following)*
      $p \leftarrow p+1$
  return $\{MDS^0, MDS^1, \ldots, MDS^l\}$

Algorithm 5.6. *Search algorithm for maximum disjunctive sets*

EXAMPLE.– The problem is defined by the following table and by $A_k = 5$:

i	1	2	3	4	5	6	7	8
$E[i]$	a	b	c	d	e	f	g	h
a_i	4	4	3	3	2	2	1	1

Three MDSs exist: $MDS^0 = \{a,b,c,d\}$, $MDS^1 = \{a,b,e\}$ and $MDS^2 = \{a,b,f\}$. Conflict resolution for this problem requires total sequencing on these three sets at least. However, there are still other conflicts to be resolved. They are characterized by sets of three or four tasks: $\{a,g,h\}$, $\{b,g,h\}$, $\{c,e,g\}$, $\{c,e,h\}$, $\{c,f,g\}$, $\{c,f,h\}$, $\{d,e,g\}$, $\{d,e,h\}$, $\{d,f,g\}$, $\{d,f,h\}$ and $\{e,f,g,h\}$.

We now present constraint propagation rules which can facilitate the analysis and resolution of the scheduling problem by relying on critical sets and maximum disjunctive sets.

5.3.2. *Deductions based on critical sets and MDSs*

The following set of rules are based on the search for sequencing conditions from which we can adjust time windows $[r_i, d_i]$ initially allocated to tasks; we will represent the current window of task i by the $[\underline{t}_i, \overline{c}_i]$ interval after propagation of conclusion of these rules.

We will start by presenting a basic rule which can be applied to any type of problem (disjunctive and cumulative) and that will delete inconsistent decisions *(forbidden precedence)* linked to the resolution of critical sets. In the case of disjunctive problems, we can use rules for much stronger conclusions. We can search for task pairs which must be sequenced. In a more general way, sequencing rules applicable to disjunctive problems attempt to demonstrate that a task must not (or must) be placed before (or after) all tasks in a set (see detection of *immediate selections* in [CAR 94] and [BRU 94] also implemented in [NUI 94], *edge-finding* rules in [APP 91], *not-descendant/not-ascendant sets* in [ERS 76a] and the *not-first/not-last* problem shown in [BAP 96]). To conclude, we propose a more general formulation of the forbidden precedence rule; this rule will sequence a pair of tasks by taking the influence of a set of tasks into consideration.

It is important to note that conclusions of rules are not explicitly stored – except for the simplest ones (type $i \prec j$), where the number remains limited – but immediately interpreted in terms of time window adjustments.

5.3.2.1. *Forbidden precedence (FP) – mandatory precedence (MP)*

The first sequencing rule (FP1), illustrated in Figure 5.2, consists of discovering *forbidden precedences*, denoted as $i \not\prec j$, i.e. pairs of tasks (*i, j*) where time windows prohibit decision $i \prec j$. Its formulation is as follows:

if $\overline{c}_j - \underline{t}_i < p_i + p_j$ **then** $i \not\prec j$ [FP1]

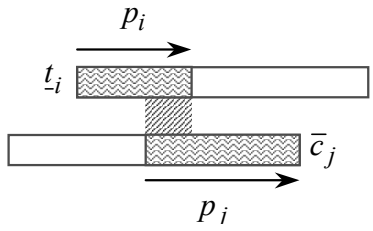


Figure 5.2. *Detection of a forbidden precedence relation*

Applied to a disjunctive pair (*i, j*) (represented by *critical* (*i, j*)), this rule shows the inconsistency of one of the possible sequences, which makes the other one *mandatory*.

if $critical(i, j)$ **and** $i \not\prec j$ **then** $j \prec i$ [MP1]

The forbidden precedence relation leads to a time window adjustment:

if $j \prec i$ **then** $\underline{t}_i \leftarrow \max\left(\underline{t}_i, \left(\underline{t}_j + p_j\right)\right)$ **and** $\overline{c}_j \leftarrow \min\left(\overline{c}_j, \left(\overline{c}_i - p_i\right)\right)$ [A1]

An inconsistency (denoted by $\perp$) appears if no sequencing is eligible:

if $critical(i,j)$ **and** $i \not\prec j$ **and** $j \not\prec i$ **then** $\perp$ [I1]

In practice, rule I1 is not required. The inconsistency linked to contradictory conclusions $i \prec j$ and $j \prec i$ (obtained by MP1) is propagated using time window adjustments (obtained by A1) and inevitably leads to a numerical inconsistency. After a finite number of adjustments, the width of one of the time windows becomes less than the task duration. Rules FP1, MP1 and A1 define a basic core for the development of a crossed propagation of time window and resource sharing constraints in the case of disjunctive problems. The search for all conclusions allowed by these rules has limited complexity. We show that the application of these rules accomplishes a *bound arc-consistency* type propagation [LHO 93, FAR 94], i.e. a propagation of disjunctive constraints in the form of window boundary adjustments, without creating holes in the domains of start times.

5.3.2.2. *Not-first (NF) – not-last (NL)*

Figure 5.3 illustrates a more general rule than FP1. This rule, denoted as NF, is based on two task subsets, $\{i\}$ and S, included in a maximum disjunctive set (must

be completely sequenced). $lst(S)$ represents an upper bound from the latest start time over all eligible S sequences. The formulation of the NF rule is as follows:

if $lst(S) - \underline{t}_i < p_i$ **then** $i \not\prec S$ [NF]

In Figure 5.3 time windows for tasks are such that any sequencing placing i before S tasks is impossible regardless of the sequence chosen for the sequencing of S.

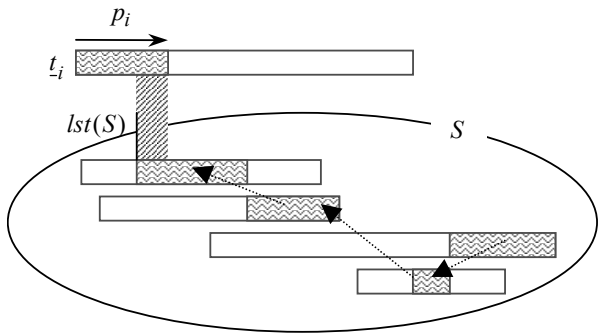


Figure 5.3. *Detection of a* $i \not\prec S$ *type condition*

Since all tasks must be sequenced, we can conclude from the NF rule that in any eligible sequencing, at least one task of S must be placed before i:

if $i \not\prec S$ **then** $\exists s \in S$ tq $s \prec i$ [MP2]

This can lead to a domain adjustment of i start time [TOR 00a]:

if $\exists s \in S$ tq $s \prec i$ **then** $\underline{t}_i \leftarrow \max\left(\underline{t}_i, \min_{s \in S}\left(\underline{t}_s + p_s\right)\right)$ [A21]

The upper bound $lst(S)$ is established by relaxing constraints of earliest start for S tasks with the help of the following algorithm:

Function LST(S)
  $E \leftarrow$ sort(S , $\overline{c}_i$, non-increasing)
  $lst \leftarrow \overline{c}_{E[1]} - p_{E[1]}$
  for k **from** 2 **to** $|E|$ **do**
    $lst \leftarrow \min(lst, \overline{c}_{E[k]}) - p_{E[k]}$
  return lst

Algorithm 5.7. *Determination of the latest start time for a set of tasks*

Rules NL, MP3 and A22 which are symmetric to NF, MP2 and A21 will not be discussed in detail. The NL rule demonstrates that a task i cannot be executed after all tasks of a S set. The relation obtained, $S \not\prec i$, will in turn make it possible to conclude on i sequencing before at least one task of S (rule MP3). The NL rule involves function $eft(S)$ which determines a lower bound of the earliest completion time over all eligible *S* sequences. Adjustment rule A22 updates time $\overline{c}_i$.

Two $O(n^2)$ versions of an algorithm to implement NF/NL rules are available in [BAP 96, TOR00a]. An $O(n\log n)$ version can now be found in [VIL 05], as well as other filtering algorithms for the implementation of the rules presented in the following (*edge-finding*). Also, it is worth reading [PER 05] where a unified framework is proposed for local adjustments.

5.3.2.3. *Non-insertable (NI) – mandatory last (ML) – mandatory first (MF)*

We are again placed in the disjunctive case. Task i is not insertable in S, a condition denoted by $i \Downarrow S$ (Figure 5.4), if no scheduling will exist accepting a sequence in the form $\alpha \prec S' \cup \{i\} \prec \beta$, $\forall \alpha, \beta \in S$, $S' = S \setminus \{\alpha, \beta\}$. Such schedules are necessarily part of the interval $[\min_{s\in S} \underline{t}_s, \max_{s\in S} \overline{c}_s]$:

if $\max_{s\in S} \overline{c}_s - \min_{s\in S} \underline{t}_s < \sum_{s\in S} p_s + p_i$ **then** $i \Downarrow S$ [NI]

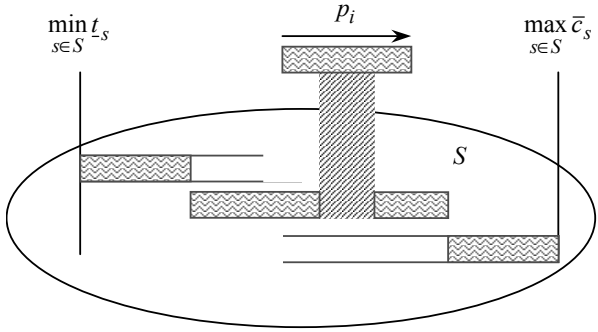


Figure 5.4. *Detection of a* $i \downarrow S$ *type condition*

Task i must therefore be executed either before or after all S tasks. If in addition one of the NL or NF rules triggers, the conclusion is very strong: by combining NI and NF (respectively NL), we can conclude that i must be in last (or first) position in relation to the tasks of S :

if $i \downarrow S$ **and** $i \not\prec S$ **then** $S \prec i$ [ML]

if $i \downarrow S$ **and** $S \not\prec i$ **then** $i \prec S$ [MF]

Since the tasks of S must be sequenced, we deduct the following adjustments:

if $S \prec i$ **then** $\underline{t}_i \leftarrow \max\left(\underline{t}_i, \mathit{eft}(S)\right)$ [A31]

if $i \prec S$ **then** $\overline{c}_i \leftarrow \min\left(\overline{c}_i, \mathit{lst}(S)\right)$ [A32]

5.3.2.4. *Generalized forbidden precedence*

One last rule enables the sequencing of two tasks in the disjunctive case. Consider two subsets S and $\{i, j\}$. To be eligible, any sequence made up of the tasks of $S \cup \{i, j\}$ meeting relation $i \prec j$ must end at date $\max_{s\in S\cup\{j\}} \overline{c}_s$ at the latest. Similarly, this type of sequence begins at date $\min_{s\in S\cup\{i\}} \underline{t}_s$ at the earliest. In this way, fulfillment duration of any sequence meeting $i \prec j$ is necessarily equal to or lower than $\max_{s\in S\cup\{j\}} \overline{c}_s - \min_{s\in S\cup\{i\}} \underline{t}_s$. Otherwise, we conclude that any sequence meeting the $i \prec j$ relation is forbidden:

$$\textbf{if} \quad \max_{s \in S \cup \{j\}} \overline{c}_s - \min_{s \in S \cup \{i\}} \underline{t}_s < \sum_{s \in S \cup \{i,j\}} p_s \quad \textbf{then} \quad i \nprec j \qquad \text{[FP2]}$$

The conclusion of FP2 can be reused (see rule MP1) to demonstrate a relation of mandatory precedence as well as possible adjustments that it implies (see rule A1). This rule, which is completely new to our knowledge, offers a generalization of FP1 (we do get to FP1 by writing $S = \varnothing$ in FP2).

5.3.3. *Deductions based on the energetic balance*

Conclusions related to energetic reasoning use resource usage balances over certain time intervals, which lead to the identification and calculation of different energies [ERS 91, LOP 91, LOP 92, LOP 96]. Over a time interval, energy can actually be provided by a resource or required by a task. In this last case, we will separately observe its minimum consumption from its maximum consumption over that time interval.

5.3.3.1. *Available energy – required energy*

Maximum available energy that a resource k provides over a $\Delta = [t_\Delta, c_\Delta]$ time interval corresponds to the product of the length of the interval by the capacity (constant) A_k of the resource:

$$W_k^\Delta = A_k \left(c_\Delta - t_\Delta\right) \qquad [5.1]$$

For a task i located in time (t_i is fixed), the energy required by (or consumption of) i over Δ, denoted w_i^Δ, is written as (Figure 5.5):

$$w_i^\Delta = a_i \max\left(0, \min\left(c_i, c_\Delta\right) - \max\left(t_i, t_\Delta\right)\right) \qquad [5.2]$$

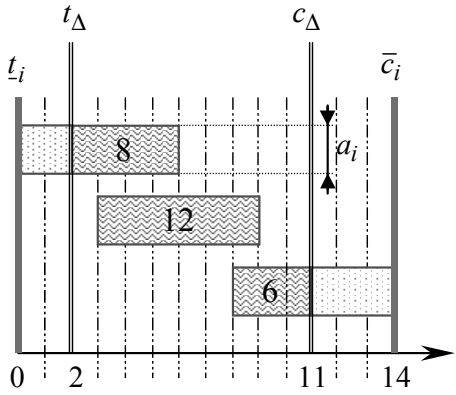


Figure 5.5. *Consumptions of task* i *(*$\left[\underline{t}_i, \overline{c}_i\right] = \left[0,14\right]$, $p_i = 6$, $a_i = 2$*) over time period* $\Delta = \left[2,11\right]$ *for different* t_i *values*

5.3.3.2. *Necessary feasibility conditions*

Now that available and required energies are defined, we can formulate a condition that is both necessary and sufficient for the availability of scheduling based on energetic balances. A given scheduling $\{t_i, \quad i = 1,\ldots,n\}$ is feasible if and only if:

$$\forall \Delta, \sum_{i \in T_k} w_i^\Delta \leq W_k^\Delta \qquad [5.3]$$

However, this condition is not directly usable because before resolution, t_i, c_i and w_i^Δ are variables. We can, however, deduct from equation [5.2] lower and upper bounds by considering the time window constraint of i. The minimum energy required by i over Δ, denoted as $\underline{w}_i^\Delta$, is calculated by examining task consumption over the interval when it is pushed back in extreme positions of its window; we also call this *compulsory consumption*:

$$\underline{w}_i^\Delta = a_i \max\left(0, \min\left(p_i, c_\Delta - t_\Delta, \underline{c}_i - t_\Delta, c_\Delta - \overline{t}_i\right)\right) \qquad [5.4]$$

For a maximum energy $\overline{w}_i^\Delta$ formula, we must replace $\underline{c}_i$ with $\overline{c}_i$ and $\overline{t}_i$ with $\underline{t}_i$ in equation [5.4].

EXAMPLE.– In the previous example (Figure 5.5), we have: $\underline{w}_i^{\Delta} = a_i\left(c_{\Delta} - \bar{t}_i\right) = 2\times 3 = 6$ and $\overline{w}_i^{\Delta} = a_i\, p_i = 2\times 6 = 12$.

Due to the compulsory consumption, we can formulate a sufficient problem inconsistency condition:

$$\textbf{if } \exists \Delta \text{ tq } W_k^{\Delta} < \sum_{i\in T_k} \underline{w}_i^{\Delta} \textbf{ then } \perp \qquad \text{[I2]}$$

5.3.3.3. *Relevant intervals for feasibility analysis*

It is possible to list in $O(n^2)$ all relevant intervals for the feasibility analysis. They can be summarized in three date sets ([LOP 91, BAP 98]):

$$O_1 = \{\underline{t}_i, i = 1,\ldots,n\} \cup \{\bar{t}_i, i = 1,\ldots,n\} \cup \{\underline{c}_i, i = 1,\ldots,n\}$$

$$O_2 = \{\bar{c}_i, i = 1,\ldots,n\} \cup \{\underline{c}_i, i = 1,\ldots,n\} \cup \{\bar{t}_i, i = 1,\ldots,n\}$$

$$O(t) = \{\underline{t}_i + \bar{c}_i - t, i = 1,\ldots,n\}$$

These three sets correspond to discontinuities of the energy consumption curve gradient when the bounds of intervals under study vary [LOP 91]. Feasibility analysis of scheduling is then carried out by studying energetic balances over $[t_{\Delta}, c_{\Delta}]$ intervals where bounds are in one of the three Cartesian products: $O_1 \times O_2$, $O_1 \times O(t), t \in O_1$ and $O(t) \times O_2, t \in O_2$.

NOTE.– In the search for a solution to a cumulative problem, an experimental study [BAP 98] of the necessary satisfiability tests and corresponding adjustments shows that energetic reasoning is too time consuming compared to existing procedures (for example [DEM 92] or [BRU 98]). We can decide to limit intervals of study to the Cartesian product of $O'_1 = \{\underline{t}_i, i = 1,\ldots,n\} \cup \{\bar{t}_i, i = 1,\ldots,n\}$ by $O'_2 = \{\bar{c}_i, i = 1,\ldots,n\} \cup \{\underline{c}_i, i = 1,\ldots,n\}$, which results in losing some conclusions.

5.3.3.4. *Adjustments*

Apart from rule I2, which concludes to global inconsistency, there are other statements of property [5.3] in the form of propagation rules implying an adjustment of domains.

For a given task i and an interval Δ, let $S_i^{\Delta} = W_k^{\Delta} - \sum_{j \in T_k \setminus \{i\}} \underline{w}_j^{\Delta}$ be the maximum available energy for executing i over Δ considering compulsory consumption of other T_k tasks:

if $\exists i, \exists \Delta$ tq $S_i^{\Delta} < \overline{w}_i^{\Delta}$ **then** there are forbidden start dates for i [FD1]

Dates leading to a larger consumption than S_i^{Δ} are inconsistent and must be deleted. Knowing that energy $\overline{w}_i^{\Delta}$ recognizes $a_i p_i$ as an upper bound, that i cannot be interrupted and that its intensity is constant during execution, we can add rule FD1. In the case where $p_i > \frac{S_i^{\Delta}}{a_i}$, a part of the task, $p_i - \frac{S_i^{\Delta}}{a_i}$, must be maintained apart from Δ. This results in a forbidden value interval for t_i: $]t_{\Delta} + \frac{S_i^{\Delta}}{a_i} - p_i, c_{\Delta} - \frac{S_i^{\Delta}}{a_i}[$. The removal of values in an initially allowed domain $[\underline{t}_i, \overline{t}_i]$, can create an increase of $\underline{t}_i$, a decrease of $\overline{t}_i$, or both (creating a hole).

Since the updating procedure can be executed in $O(1)$ and given $O(n^2)$ intervals to consider for n tasks to be updated, all updating resulting from the application of rule FD1 can be carried out in $O(n^3)$.

5.3.3.5. *Sequencing relations*

Another application of energetic reasoning involves the sequencing of two tasks. For a given pair of tasks (i, j) and an interval Δ, let $S_{(i,j)}^{\Delta} = W_k^{\Delta} - \sum_{l \in T_k \setminus \{i,j\}} \underline{w}_l^{\Delta}$ be the maximum available energy to execute i and j over Δ considering compulsory consumption of other tasks of T_k. We have the following rule:

if $\exists i, j, \Delta$ tq $S_{(i,j)}^{\Delta} < \overline{w}_i^{\Delta} + \overline{w}_j^{\Delta}$

then there are forbidden pairs of execution dates for i and j [FD2]

In other words, if i and j both consume over Δ, they must be separated by a minimum distance. However, we cannot specify in which relative position i and j are. This rule does not allow a direct interpretation in terms of time window adjustments. For this purpose, we must consider additional hypotheses.

First, we must limit ourselves to disjunctive problems and choose intervals Δ such that consumptions of both tasks when $i \prec j$ are maximal, for example $\Delta = [\underline{t}_i, \overline{c}_j]$; we can then conclude to a new forbidden precedence relation (Figure 5.6):

if $\overline{c}_j - \underline{t}_i < p_i + p_j + \sum_{l \in T_k \setminus \{i,j\}} \underline{w}_l^{\Delta}$ **then** $i \not\prec j$ [FP3]

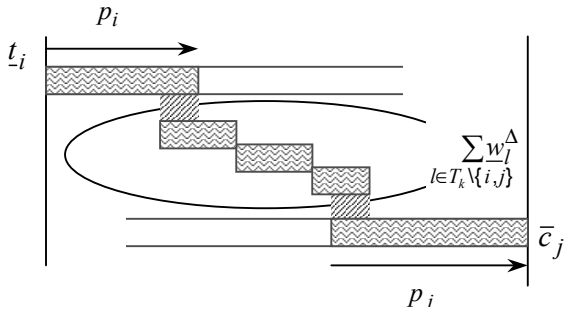


Figure 5.6. *New forbidden precedence relation*

FP3 dominates FP1. It can be triggered in cases where FP1 would not conclude. On the other hand, if FP1 is triggered, FP3 will as well.

On the other hand, by choosing intervals of the form $\Delta = \left[\min_{J_i} \underline{t}_j , \max_{J_i \cup \{i\}} \overline{c}_j \right]$ with $J_i = \{j = 1,\ldots,n \,/\, j \neq i, \quad \underline{t}_i < \underline{t}_j < \overline{c}_i\}$, another statement of rule FD2 helps in finding the domain adjustment produced by rule A32 (sequencing of i before all tasks of J_i).

The energetic reasoning provides rules covering powerful and well-known propagation rules (disjunction pairs, immediate selections, not-descendant/not-ascendant sets). It is very general, since it also makes it possible to provide rules deducing some forbidden start times (in its most general expression, rule FD1 leads to the creation of "holes" in the domain of t_i). This has been proven to be of great interest for various scheduling problems (see for example [TER 06]).

5.4. Integration of propagation techniques in search methods

The use of constraint propagation mechanisms within a backtracking algorithm (see section 5.1.2) limits the search space for solution. However, a compromise still needs to be made. In fact, powerful propagation mechanisms in terms of solution space reduction can simplify the resolution (it can even in certain cases be carried out with no backtracking), but can sometimes be expensive in terms of calculation time. In addition, absence of propagation mechanisms leads to useless explorations of solution space and results in numerous backtrackings. If we examine the different types of propagation rules previously described, we observe that these rules manipulate the main variables of the problem $\{t_i\}$ and/or $\{c_i\}$ where they try to filter the domain, symbolic relations (type $i \prec j$), but also, in the case of energetic reasoning, intermediate variables (quantities w_i^{Δ}).

A first resolution algorithm consists of limiting variables used only for temporal variables t_i and to progressively instantiate them. At each step, a decision is propagated as far as possible before moving on to the following variable. The resulting search tree contains as many levels as there are temporal variables. In each node, a variable is selected (the order of variable choices is not set beforehand); we create as many successor nodes as there are possible values in the domain of the chosen variable.

A second resolution algorithm favors the resolution of resource utilization conflicts and is meant to resolve the problem by sequencing conflicting tasks. In this case, the search tree developed is different from the previous one. It is based on sequencing variables and includes as many levels as there are critical sets. For example, in the disjunctive case a binary x_{ij} variable represents possible sequences between two conflicting tasks i and j ($x_{ij}=1 \Leftrightarrow i \prec j$ and $x_{ij}=0 \Leftrightarrow j \prec i$).

Other resolution schemas can be considered. In the disjunctive case for example, we can consider task ranks in the sequence on the machine as a variable set.

In the following, we present control strategies for the development of a backtracking algorithm based on propagation mechanisms. These strategies are explained through different choices involving backtracking techniques, variable and value selection heuristics and the selection and control of propagation rules. One last point is dedicated to the use of this algorithm for the search for optimal solutions.

5.4.1. *General improvement techniques of chronological backtracking*

In the general resolution schema with chronological backtracking, step l of the resolution is characterized by the search for a consistent instantiation $x_l \leftarrow v_l$ considering already completed instantiations $\{x_1 \leftarrow v_1, x_2 \leftarrow v_2, \ldots, x_{l-1} \leftarrow v_{l-1}\}$. If there is some inconsistency at step l (no value can be attributed to x_l), the last completed instantiation ($x_{l-1} \leftarrow v_{l-1}$) is called into question, although this is not necessarily involved in the failure detected.

Different "intelligent" backtracking techniques can be imagined [DEC 90, TSA 93, DEC 03]. The *backjumping* technique consists of returning to the previous most recent step k ($k \leq l-1$), such that there is at least one constraint involving both variables x_l and x_k. This technique is all the more efficient the lower the number and arity of constraints (sparse constraint graph not very *dense*). Unfortunately, in the case of scheduling problems, resource sharing constraints create a strong pairing between temporal variables, in particular cumulative constraints linking temporal variables in the set of tasks sharing a resource. This technique can become appropriate if we can decompose the scheduling problem in order to decrease the constraint arity involved in sub-problems.

A second technique is based on recording failures encountered during resolution in the constraint form in order to avoid reproducing them. This technique, called *constraint recording* (or *nogood constraints*), consists of storing causes for a failure during resolution by explaining implicit constraints in the initial problem formulation. In order to do this, when an inconsistency appears for the instantiation of a variable x_l, we search among all previous instantiations $\{x_1 \leftarrow v_1, x_2 \leftarrow v_2, \ldots, x_{l-1} \leftarrow v_{l-1}\}$, the subsets liable to be the cause of this failure. These subsets are then stored in the form of constraints to keep the failure from occurring again. For efficiency purposes, we generally only store constraints with low arity (one or two variables).

This *constraint recording* technique is more powerful than *backjumping* for dense constraint graphs. On the other hand, only storing low arity constraints is limiting for scheduling problems for which pairing between tasks is high since the reasons for a failure during resolution cannot be translated in the form of binary constraints [SAD 96].

5.4.2. *Heuristics for variable and value ordering*

Variable and value ordering heuristics have an important impact on the performance of solution search procedures [SMI 93, BAP 95, SAD 96]. These heuristics can be static or dynamic (based on propagation results, for example). As an example, general variable ordering heuristics select the most constrained variable first (the one for which there is the least choice of values) or the most restrictive variable (the one linked to the highest number of other variables).

In the disjunctive problem case (job shop) for which problem resolution is based on the search for a sequencing of disjunctive tasks, Smith and Cheng define a series of heuristics based on temporal flexibility associated with different sequencing decisions [SMI 93]. These are task selection heuristics based on the temporal slacks linked to a sequencing: $\text{slack}(i \prec j) = \overline{c}_j - \underline{t}_i - (p_i + p_j)$; they are associated with rules of conflict-based constraint analysis, notably the MP rule. This heuristic selects the pair of tasks to sequence (i,j); it is identified by the pair where one of the sequences gives the minimum margin:

$$\min(\text{slack}(i \prec j), \text{slack}(j \prec i)) = \min_{u,v}\{\min[\text{slack}(u \prec v), \text{slack}(v \prec u)]\} \qquad [5.5]$$

for all unsequenced pairs (u, v).

For this selected pair (i, j), we choose the sequencing with the highest slack, for example $j \prec i$ if $\text{slack}(j \prec i) > \text{slack}(i \prec j)$.

Experienced without allowing backtracking (which does not guarantee the production of a feasible solution), this heuristic is called "min-slack/max-slack" and obtains very good results compared to approaches implementing *backtracking* techniques either chronological or intelligent (for 60 randomly generated problems, the heuristic, programmed in C, resolves 56 problems in an average CPU time of 0.2 s per problem resolved on a Decstation 5000[3]). The results are also far superior to those obtained by more traditional priority rules used for tardiness minimization (*Earliest Due Date, Cost OVER Time, Apparent Tardiness Cost*; see Chapter 6).

In a more general way, in the application of rules involving sets of tasks, certain tasks are sorted (rules MF and ML), while others are not (including those handled by rules NF and NL). For these, an indeterminism remains and a choice must be made

3 The authors propose refining the method by introducing a second heuristic for sequencing tasks with equal priority by min-slack/max-slack. They then resolve all problems generated in a short time period (0.3 seconds per problem resolved).

for the tasks to select and the way in which to place them for efficient resolution. By the application of the NF and NL rules, we identify tasks which can be first or last. Baptiste *et al*. [BAP 95] propose different heuristics according to whether we prefer to always select a first task, a last task or decide to apply dynamic criteria such as the number of tasks which can be executed first or last. In this way, they indicate that it is better to select a task belonging to the smallest possible-first or possible-last group, in order to limit the number of branching in the search tree. For the selection of a possible first task, a rule based on earliest start time gives the priority on the selection of tasks; latest start time intervenes to break ties (the global rule is called EST-LST). For a possible-last task, the LFT-EFT rule (latest finishing time-earliest finishing time) is the rule applied.

5.4.3. *Strategies for applying propagation rules*

Propagation mechanisms can be applied during resolution when variables are instantiated in order to prune the search tree. The simplest application principle of propagation mechanisms is *forward checking* consisting of propagating the instantiation value which was only performed on variables linked to the current variable by a constraint (not yet instantiated). Propagating each instantiation to all variables of the problem can also be considered. In this case, we speak of *full lookahead* or *Maintaining Arc-Consistency (MAC)*. In the CSP domain, propagation rules considered for resolution algorithms are only rarely meant for the most detailed explanation of all constraints of a problem, except for simple problems for which this explanation guarantees a resolution algorithm with no backtracking. In practice, for temporal problems, this means that we do not try to explain new constraints as can be the case with the Floyd-Warshall algorithm and that we limit ourselves to the Bellman-Ford algorithm which restricts variable domains.

In a shop scheduling problem, it is important to develop a relevant strategy for the application of propagation rules. The strategy retained in [BAP 95] and [TOR 00a] consists of selecting the resource with the lowest slack, where a resource slack is defined by the difference between offer and demand for the use of this resource in the time window of each unsequenced task. The propagation rules are then applied on this resource until there is no longer a deduction, before moving on to the next resource in the decreasing slack order.

5.4.4. *Use of a backtracking algorithm*

Criteria optimization can be carried out with the help of a binary search procedure using a backtracking algorithm. For example, for makespan minimization, the optimization procedure determines an initial scheduling horizon H located in

the middle of trivial lower and upper bounds deducted by a (meta)heuristic. This horizon value is used as scheduling makespan and restricts completion times for the different tasks of the problem. For this horizon H, the optimization procedure uses a backtracking algorithm to determine a solution. If this algorithm does not detect global inconsistency of the problem over horizon H, i.e. if a solution is found, the value of H is then used as the new upper bound; otherwise the lower bound becomes $H+1$. The optimization procedure continues by determining a new H value by dichotomy. The *optimum* is reached when the lower and upper bounds are equal.

Based on this principle, we can cite results obtained for the resolution of cumulative problems ([BAP 98], section C.3). These results show that propagation rules based on energetic reasoning are all the more powerful as the problem is highly cumulative (i.e. with a low disjunction pair percentage compared with the total number of pairs of tasks). For 31 highly cumulative problems generated by Baptiste and Le Pape (disjunction percentage of 0.33), the energetic reasoning divides the number of backtracks by 45 and the CPU time necessary to find an optimal solution by 7. However, for Alvarez problems (disjunction percentage of 0.82), Patterson (0.67) or Kolisch (0.56), the results may seem disappointing. In addition, excellent results have been obtained for the hybrid flow shop resolution (see Chapter 9).

5.5. Extensions

5.5.1. *Preemptive problems*

A hypothesis which may be interesting to consider is that linked to the uninterruptibility of tasks. We often estimate that this *preemption* should not be considered in production and that it specifically addresses computer scheduling problems. We can however consider preemption to be the relaxation of a non-preemptive problem making it possible to implement new methods of reasoning. New necessary conditions for the existence of a feasible solution for cumulative and preemptive scheduling are established in [BAP 98, BAP 01]. These conditions involve the resolution of a *partially* or *fully elastic*[4] problem, a definition which is based on the notion of energy from section 5.3.3 extended to the preemptive case.

4 In a fully elastic problem, we can change the task at will (since the energy required remains constant) as long as we do not violate resource capacity. In a partially elastic problem, we restrict the previous change by imposing a higher energy consumption limit over a given time interval.

When we try to minimize total time, considering preemption leads to adjustment formulae of the earliest completion time for tasks (instead of the earliest start time). The FD1 conclusion (see section 5.3.3.4) would result in the new constraint:

$$c_i \geq c_\Delta - \frac{S_i^\Delta}{a_i} + p_i$$

In [BAP 01], we find:

$$c_i \geq c_\Delta - \frac{S_i^\Delta}{a_i} + \max\left(0, \min\left(p_i, c_\Delta - t_\Delta, \underline{c}_i - t_\Delta\right)\right).$$

This last formula is more general since it ensures a lower bound of the completion time in the preemptive case. In the non-preemptive case, it is lower since the term $\min\left(p_i, c_\Delta - t_\Delta, \underline{c}_i - t_\Delta\right)$ is always less than or equal to p_i. The same earliest start time is found in several cycles by applying the rule several times over the same Δ interval.

In the partially elastic case, it has been shown that for the determination of intervals Δ, the idea is to consider the Cartesian product of the earliest start times and latest completion times sets to completely obtain all relevant intervals of study in order to decide on the existence of a feasible schedule.

5.5.2. *Consideration of allocation constraints*

Problem statement

In this section, scheduling problems considered in section 5.1.3 are extended to problems for which assignment of tasks to resources is not set beforehand. A set of resources $\mathsf{R}_i \subseteq \mathsf{R}$, able to complete the task, is associated with each task i. For each k resource of R_i, the duration (processing time) of task i is denoted by $p_{i,k}$ and can be variable: $p_{i,k} \in [\underline{p}_{i,k}, \overline{p}_{i,k}]$, or set: $p_{i,k} = \underline{p}_{i,k} = \overline{p}_{i,k}$. As with parallel machine problems, we can separate problems according to resource characteristics. When resources are identical, the duration of a task does not vary according to the resource to which it is assigned. When resources are uniform, the duration varies proportionally based on the allocated resource; then on independent resources, the duration of a task is ordinary according to the resource used for its execution.

Efficient algorithms exist for processing scheduling and allocation problems separately, but they do not guarantee finding a relevant solution for the complete problem. In this context of integrated resolution of scheduling and allocation problems, a certain number of studies were carried out. We can for example cite studies on flow shops with allocation constraints, also called hybrid flow shops (see Chapter 9) and job shops with allocation constraints also called flexible job shops [BRA 93, HUR 94]. Results on the complexity of such problems are presented in [BRU 97]. More general scheduling problems, including allocation constraints, have also been studied, for example multiresource scheduling problems where resources can be cumulative and production routes can be non-linear, and considering (or not) setup times [KOL 95, BIL 96, DAU 98, ART 99], multiresource job shop problems [NUI 94], or flexible job shop problems integrating transportation constraints [PAU 95]. In most cases, the methods used are exact tree search methods or approximate methods such as metaheuristics.

To our knowledge, the only studies based on constraint propagation techniques for scheduling and allocation problems are those from Nuijten, and Huguet and Lopez. In [NUI 94], with each task and allocation possibility, we associate a time window. Developed constraint propagation techniques are based on traditional scheduling propagation mechanisms and make it possible to restrict task time windows; this can lead to the removal of allocation choices. A propagation rule dedicated to allocation constraints is also implemented; it is based on a resource aggregation to which different tasks can be allocated to a cumulative resource. For example, consider three tasks, i, j, k which can be assigned either on resource r_1 or on resource r_2. In the disjunctive case, at most two tasks may be executed simultaneously by set of resources $\{r_1, r_2\}$. This set of resources is thus seen as an aggregated resource with a capacity of 2 and mechanisms of cumulative resource propagation based on NF and NL rules (see section 5.3) are proposed. In [HUG 99], allocation choices are interpreted in terms of possible task duration constraints. This interpretation brings the scheduling and allocation problem back to a scheduling problem in which task durations are variable. Constraint propagation mechanisms similar to those used in [NUI 94] are implemented. In addition, specific rules based on energetic reasoning are proposed for the propagation of allocation constraints [HUG 00].

5.6. Conclusion

The object of this chapter was to present major constraint propagation techniques for the resolution of scheduling problems. These techniques characterize the set of feasible solutions and offer a formal context for addressing these problems from a decision support context.

We first focused on the purely temporal aspect of the scheduling problem. General results from the study of constraint satisfaction problems were exposed. For simple temporal constraints, there are complete propagation algorithms with acceptable complexity based on the search for the longest paths in the constraint graph. When we must deal with more sophisticated temporal constraints (interval disjunction), we find incomplete reasoning.

We then considered problems in which tasks must respect temporal and resource sharing constraints simultaneously. We have shown a range of constraint analysis rules based on conflicts between tasks for the use of resources and rules involving energetic balances, where satisfaction is the source of propagations over the absolute or relative positioning of tasks. We have studied extensions of propagation rules for problems where task interruption is authorized or assignment of tasks to resources is not entirely set.

Finally, we discussed control strategies for the development of a backtracking algorithm involving propagation mechanisms. This can help in the quick discovery of global inconsistency in the problem formulation, to simplify the resolution and to search for an optimal solution.

To conclude, we should note that this presentation is not exhaustive. For example, there are powerful propagation techniques for the processing of disjunctive scheduling problems. They are known as *global operations* or *shaving* [CAR 94, MAR 96]. A local constraint is raised (instantiation of a start time or sequencing decision) and corresponding adjustments are propagated over the whole problem. If inconsistency is detected, the negation of the constraint raised must be checked and is thus added to the problem definition. We can consider a hybrid use of *shaving* type propagation techniques and local search procedures (notably the Tabu method) to enable the efficient extraction of local optimas and solve large job-shop problems (15 tasks-10 machines and beyond) [TOR 00b].

5.7. Bibliography

[ALL 83] ALLEN J.F., "Maintaining knowledge about temporal intervals", *Communications of the ACM*, vol. 26, p. 832-843, 1983.

[APP 91] APPLEGATE D. and COOK W., "A computational study of the job-shop scheduling problem", *ORSA Journal on Computing*, vol. 3, no. 2, p. 149-156, 1991.

[ART 99] ARTIGUES C. and ROUBELLAT F., "Real time multi-resource shop floor scheduling with setup times", *International Conference on Industrial Engineering and Production Management (IEPM-99)*, vol. 2, p. 127-136, 1999.

[BAP 95] BAPTISTE P., LE PAPE C. and NUIJTEN W., "Constraint-based optimisation and approximation for job-shop scheduling", *14th IJCAI*, Montreal, Canada, 1995.

[BAP 96] BAPTISTE P. and LE PAPE C., "Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling", *UK Planning and Scheduling SIG Meeting*, Liverpool, 1996.

[BAP 98] BAPTISTE P., A theoretical and experimental study of resource constraint propagation, Doctoral Thesis, Compiègne University of Technology, 1998.

[BAP 01] BAPTISTE P., LE PAPE C. and NUIJTEN W., *Constraint-based Scheduling*, Kluwer Academic Publishers, Boston, 2001.

[BAR 88] BARTUSCH M., MÖHRING R.H. and RADERMACHER F.J., "Scheduling project networks with resource constraints time windows", *Annals of Operations Research*, vol. 16, p. 201-240, 1988.

[BEL 82] BELLMAN R., ESOGBUE A.O. and NABESHIMA I., *Mathematical Aspects of Scheduling and Applications*, Pergamon Press, Oxford, 1982.

[BEL 89] BEL G., BENSANA E., DUBOIS D., ERSCHLER J. and ESQUIROL P., "A knowledge-based approach to industrial job-shop scheduling", in *Knowledge-based Systems in Manufacturing*, A. Kusiak (ed.), p. 207-246, Taylor & Francis, 1989.

[BIL 96] BILLAUT J.-C. and ROUBELLAT F. "Characterization of a set of schedules in a multiresource context", *Journal of Decision Systems*, vol. 5, no. 1-2, p. 95-109, 1996.

[BRA 93] BRANDIMARTE P., "Routing and scheduling in a flexible job shop by tabu search", *Annals of Operations Research*, vol. 41, p. 157-183, 1993.

[BRU 94] BRUCKER P., JURISH B. and KRÄMER A., "The job shop scheduling problem and immediate selections", *Annals of Operations Research*, vol. 50, p. 73-114, 1994.

[BRU 97] BRUCKER P., JURISH B. and KRÄMER A., "Complexity of scheduling problems with multipurpose machines", *Annals of Operations Research*, vol. 70, p. 57-73, 1997.

[BRU 98] BRUCKER P., KNUST S., SHOO A. and THIELE O., "A branch and bound algorithm for the resource-constrained project scheduling problem", *European Journal of Operational Research*, vol. 107, p. 272-288, 1998.

[BRU 02] BRUCKER P., "Scheduling and constraint propagation", *Discrete Applied Mathematics*, vol. 123, p. 227-256, 2002.

[CAR 84] CARLIER J., Problèmes d'ordonnancement à contraintes de ressources: algorithmes et complexité, State Doctoral Thesis, Paris VI University, 1984.

[CAR 94] CARLIER J. and PINSON E., "Adjustment of heads and tails for the job-shop problem", *European Journal of Operational Research*, vol. 78, p. 146-161, 1994.

[CAS 95] CASEAU Y. and LABURTHE F., "Improving branch and bound for jobshop scheduling with constraint propagation", *8th Franco-Japanese – 4th Franco-Chinese Conference on Combinatorics and Computer Science*, M. Deza, R. Euler and Y. Manoussakis (eds.), Brest, France, 1995.

[DAU 98] DAUZÈRE-PÉRÈS S., ROUX W. and LASSERRE J.B., "Multi-resource shop scheduling with resource flexibility", *European Journal of Operational Research*, vol. 107, p. 289-305, 1998.

[DAV 87] DAVIS E., "Constraint propagation with interval labels", *Artificial Intelligence*, vol. 32, no. 3, p. 281-331, 1987.

[DEC 90] DECHTER R., "Enhancement schemes for constraint processing: backjumping, learning and cutset decomposition", *Artificial Intelligence*, vol. 41, p. 273-312, 1990.

[DEC 91] DECHTER R., MEIRI I. and PEARL J., "Temporal constraint networks", *Artificial Intelligence*, vol. 49, p. 61-95, 1991.

[DEC 03] DECHTER R. *Constraint Processing*, Morgan Kaufmann, San Francisco, 2003.

[DEM 92] DEMEULEMEESTER E. and HERROELEN W., "A branch and bound procedure for the multiple resource-constrained project scheduling problem", *Management Science*, vol. 38, no. 12, p. 1803-1818, 1992.

[DIN 90] DINCBAS M., SIMONIS H. and VAN HENTENRYCK P., "Solving large combinatorial problems in logic programming", *Journal of Logic Programming*, vol. 8, no. 1-2, p. 74-94, 1990.

[DOR 02] DORNDORF U., *Project Scheduling with Time Windows*, Physica-Verlag, Heidelberg, 2002.

[ERS 76a] ERSCHLER J., Analyse sous contraintes et aide à la décision pour certains problèmes d'ordonnancement, State Doctoral Thesis, Paul Sabatier University, Toulouse, 1976.

[ERS 76b] ERSCHLER J., ROUBELLAT F. and VERNHES J.-P., "Finding some essential characteristics of the feasible solutions for a scheduling problem", *Operations Research*, vol. 24, no. 4, p. 774-783, 1976.

[ERS 80] ERSCHLER J., ROUBELLAT F. and VERNHES J.-P., "Characterizing the set of feasible sequences for n jobs to be carried out on a single machine", *European Journal of Operational Research*, vol. 4, no. 3, p. 189-194, 1980.

[ERS 91] ERSCHLER J., LOPEZ P. and THURIOT C., "Raisonnement temporel sous contraintes de ressources et problèmes d'ordonnancement", *Revue d'Intelligence Artificielle*, vol. 5, no. 3, p. 7-36, 1991.

[ESQ 87] ESQUIROL P., Règles et processus d'inférence pour l'aide à l'ordonnancement de tâches en présence de contraintes, Doctoral Thesis, Paul Sabatier University, Toulouse, 1987.

[FAR 94] FARGIER H., Problèmes de satisfaction de contraintes flexibles: application à l'ordonnancement de production, Doctoral Thesis, Paul Sabatier University, Toulouse, 1994.

[FOX 83] FOX M.S., Constraint-directed search: A case study of job-shop scheduling, PhD Thesis, Carnegie Mellon University, 1983.

[HUG 99] HUGUET M.-J. and LOPEZ P., "An integrated constraint-based model for task scheduling and resource assignment", *First International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'99)*, Ferrara, Italy, 1999.

[HUG 00] HUGUET M.-J. and LOPEZ P., "Mixed task scheduling and resource allocation problems", *Second International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'2000)*, Paderborn, Germany, 2000.

[HUR 94] HURINK J., JURISCH B. and THOLE M., "Tabu search for the job shop scheduling problem with multipurpose machines", *OR Spektrum*, vol. 15, p. 205-215, 1994.

[KOL 95] KOLISCH R., "Project scheduling under resource constraints", *Production and Logistics*, Physica Verlag, Heidelberg, 1995.

[LEP 91] LE PAPE C., Constraint propagation in planning and scheduling, Technical report, CIFE, Stanford University, 1991.

[LEP 94] LE PAPE C., "Constraint-based programming for scheduling: an historical perspective", Working Notes of the Operations Research Society Seminar on Constraint Handling Techniques, London, 1994.

[LHO 93] LHOMME O., "Consistency techniques for numeric CSPs", *13th IJCAI*, p. 232-238, Chambéry, France, 1993.

[LOP 91] LOPEZ P., Approche énergétique pour l'ordonnancement de tâches sous contraintes de temps et de ressources, Doctoral Thesis, Paul Sabatier University, Toulouse, 1991.

[LOP 92] LOPEZ P., ERSCHLER J. and ESQUIROL P., "Ordonnancement de tâches sous contraintes: une approche énergétique", *Automatique-Productique-Informatique Industrielle*, vol. 26, no. 5-6, p. 453-481, 1992.

[LOP 96] LOPEZ P. and ESQUIROL P., "Consistency enforcing in scheduling: A general formulation based on energetic reasoning", *5th International Workshop on Project Management and Scheduling*, p. 155-158, Poznan, Poland, 1996.

[LOP 03] LOPEZ P., Approche par contraintes des problèmes l'ordonnancement et d'affectation : Strucures temporelles et mécanismes de propagation, Habilitation à Diriger des Recherches, Institut National Polytechnique de Toulouse, 2003.

[MAR 96] MARTIN P. and SCHMOYS D.B., "A new approach to computing optimal schedules for the job-shop scheduling problem", *5th Conference on Integer Programming and Combinatorial Optimization*, Vancouver, British Columbia, 1996.

[MEI 96] MEIRI I., "Combining qualitative and quantitative constraints in temporal reasoning", *Artificial Intelligence*, vol. 87, p. 343-385, 1996.

[NUI 94] NUIJTEN W., Time and resource constrained scheduling – a constraint satisfaction approach, Doctoral Thesis, Eindhoven University of Technology, 1994.

[PAU 95] PAULLI J., "A hierarchical approach for the FMS scheduling problem", *European Journal of Operational Research*, vol. 86, p. 32-42, 1995.

[SAD 96] SADEH N. and FOX M.S., "Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem", *Artificial Intelligence*, vol. 86, p. 1-41, 1996.

[SCH 97] SCHWALB E. and DECHTER R., "Processing disjunctions in temporal constraint networks", *Artificial Intelligence*, vol. 93, p. 29-61, 1997.

[SCH 98] SCHWALB E. and VILA L., "Temporal Constraints", *Constraints: An International Journal*, vol. 2, p. 129-149, 1998.

[SMI 93] SMITH S. and CHENG C., "Slack-based heuristics for constraint satisfaction scheduling", *AAAI-93* (American Association for Artificial Intelligence), p. 139-144, Washington, 1993.

[TER 06] TERCINET F., NERON E. and LENTE C., "Energetic reasoning and bin-packing problem, for bounding a parallel machine scheduling problem", *4OR*, vol. 4, p. 297-317, 2006.

[TOR 00a] TORRES P. and LOPEZ P., "On not-first/not-last conditions in disjunctive scheduling", *European Journal of Operational Research*, vol. 127, no. 2, 2000.

[TOR 00b] TORRES P. and LOPEZ P., "Overview and possible extensions of shaving techniques for Job-Shop problems", *Second International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'2000)*, Paderborn, Germany, 2000.

[TSA 93] TSANG E.P.K., *Foundations of Constraint Satisfaction,* Academic Press Ltd., London, 1993.

[VAN 94] VAN HENTENRYCK P., "Scheduling and packing in the constraint language CC(FD)", in *Intelligent Scheduling*, M. Zweben and M.S. Fox (eds.), Morgan Kaufmann, San Francisco, 1994, p. 137-167.

[VIL 86] VILAIN M. and KAUTZ H., "Constraint propagation algorithms for temporal reasoning", *AAAI-96* (American Association for Artificial Intelligence), p. 377-382, Philadelphia, PA, 1996.

[VIL 05] VILIM P., BARTAK R., CEPEK O., "Extension of $O(n\log n)$ filtering algorithms for the unary resource constraint to optional activities", *Constraints: An International Journal*, vol. 10, no. 4, 2005.

Chapter 6

Simulation Approach

6.1. Introduction

This chapter focuses on presenting the "simulation" approach, a scheduling problem resolution approach widely used in a number of manufacturing software packages dedicated to this type of problem. This approach is part of the class of "heuristic" methods. It considers a large number of constraints but not all those encountered in the scheduling problem resolution. It does not directly consider global criteria. It is configurable and we will see that it is possible to characterize parameter choices which are best adapted to certain situations and to obtain "good" solutions in most cases. It is relatively easy to implement. We may consider its iterative use for improving the quality of the proposed solution or to evaluate the variation effects of certain characteristics of the problem over results obtained.

This chapter is made up of five parts:

– in section 6.2, we quickly present manual resolution procedures still used and their possible evolutions considering results presented in the previous chapters;

– section 6.3 focuses on presenting discrete event simulation: the different modeling elements inherent to this technique and corresponding resolution methods;

– in section 6.4, we show how discrete event simulation can be used for building projected schedules, or for real-time decision support;

– section 6.5 discusses priority rules: classifications, relations between (local) rules and (global) performance criteria, support for the choice of rules;

Chapter written by Gérard BEL and Jean-Bernard CAVAILLÉ.

– in section 6.6, we quickly present computer tools for the implementation of such an approach.

6.2. Heuristic resolution (greedy) procedures

6.2.1. *Limits of the basic method*

Local approach

Chapter 2 has shown that we know methods for determining an optimal schedule in terms of different types of criteria for the one-machine scheduling problem. These methods are quite simple to implement. In order to determine this optimal schedule, it is simply necessary to have access to certain characteristics of jobs to schedule and to be able to classify these jobs with the help of indicators calculated from these characteristics.

This apparent ease of optimal scheduling determination can encourage the generalization of this type of approach to job shop scheduling calculation containing several machines; we use one of the resolution methods adapted to the one-machine problem to build independently the schedule for each machine in the shop. The implementation of such a policy raises three problems:

– the algorithms used for one-machine problems presume that we know the *availability* dates r_i of job i (we often presume that all jobs are available at date 0). However, the availability date of a job for machine M_k depends on the date where this job will have finished previous operations from other machines. This date depends on the schedule made on these machines. This schedule is not known at first; it must be determined. We can see that this approach can only be implemented by an iterative method which will solve one-machine scheduling problems in a certain order. In the case of job shops, it is difficult to easily determine this order:

– certain constraints are difficult to consider for machine M_k:

- either because they involve a group of machines (production routes, for example),

- or because they are relative to a given machine. For example, the delivery date of a job is the date of the end of operations on the last machine and it is difficult to ensure satisfaction of this constraint with other machines;

– optimality of the solution, in the sense of criteria, is not absolutely ensured because a global *optimum* is not obtained by a sum of local *optima* except in specific cases.

Constraint analysis

We have seen in the previous chapter how constraint analysis will restrict value domains of the different operation t_i start dates. This analysis is all the more powerful as it is more widespread. However, the more widespread it is, the more difficult it becomes. The minimal version of this propagation may be only local. In this case, for each scheduling decision made for operation *o(i, j)* on machine M_k, we only carry out a local propagation on the machine M_k and job *i* involved. We prohibit on machine M_k the time slot reserved for this operation *o(i, j*) and we deduct the earliest start time for the next operation *o(i, j + 1)* for this job.

6.2.2. *Manual development procedures of projected scheduling*

Part manufacturing scheduling problems (that we will call "job" in the rest of the chapter for homogenity purposes) have been discussed for a long time and we did not wait for calculators to solve them. They are very often "manually" resolved using "mural planners" containing as many lines (or rulers) as there are machines M_k on which we want to schedule *i* jobs to execute (Gantt chart). Rectangular cardboard pieces represent operations *o(i, j)* (operation *j* for job *i*, *i* production route element) to be scheduled and the length of these rectangles is proportional to p_{ij} duration of operation *o(i, j)* in compliance with a time scale represented based on the diagram's *x* abscissa axis. Scheduling software displays show this type of representation.

It is a specialized operator who carries out scheduling by placing rectangles on the different diagram lines one by one. In order to do this, he uses a systematic procedure followed by several adjustments when the first solution does not suit him. This method of placing the operations one by one to develop a schedule is at the basis of several heuristic resolution methods for these problems.

6.2.3. *Job placement procedure*

The systematic procedure that has been used for the longest time is the "job placement" procedure. The resulting construction method consists of:

– classifying the different jobs *i* (made up of a certain number of operations *o(i, j)*) to be scheduled according to a sequence chosen by the user. This classification *C* will be very important for the quality of the result;

– starting an iterative job placement procedure. In order to do this:

- we choose the first job i_0 in classification *C*,

- we place operations for this job in relation to operations already sequenced (free time slots on machines involved). There can be different operation placement policies (earliest, latest, etc.),

- we choose a new job in the classification and we iterate, etc..

This procedure is particularly interesting for manual implementation because it makes it possible to consider operation sequencing constraints very simply.

Once this schedule is finished, we can measure the degree of satisfaction of certain constraints which have not been directly considered in the placement procedure. This is the case with delivery date satisfaction; in fact, these dates associated with the different jobs are not directly considered during scheduling development. We have, however, been able to consider these delivery dates for the classification for these jobs. For example, we have been able to classify jobs by increasing delivery dates. This certainly does not guarantee that all delivery dates will be satisfied as the following example shows.

6.2.4. *Example*

A job shop contains three machines M_1, M_2 and M_3. In this shop, three types of jobs A, B and C are produced. The corresponding part routings are:

– A:

- o(A,1) on M_1; for $p_{A1} = 4$,

- o(A,2) on M_2; for $p_{A2} = 3$,

- o(A,3) on M_3; for $p_{A3} = 5$,

requested delivery date: $d_A = 17$;

– B:

- o(B,1) on M_3; for $p_{B1} = 3$;

- o(B,2) on M_2; for $p_{B2} = 5$;

- o(B,3) for M_3; for $p_{B3} = 3$;

requested delivery date: $d_B = 16$;

– C:

- o(C,1) on M_2; for $p_{C1} = 5$;

requested delivery date: $d_C = 10$.

We wish to execute projected job scheduling in this job shop. To use the placement method, we first classify jobs, for example, in decreasing delivery date sequence. The classification is as follows: {C, B, A}. We then consider jobs in this sequence and we place operations for each one in the sequence given by the production route.

The sequence of operation placement is: o(C,1) on M_2, o(B,1) on M_3, o(B,2) on M_2, o(B,3) on M_3 , o(A,1) on M_1, o(A,2) on M_2 and o(A,3) on M_3. We obtain the following schedule:

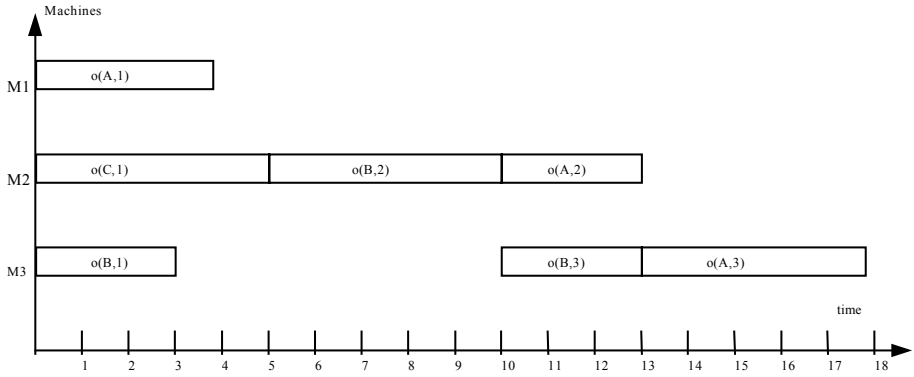


Figure 6.1. *Schedule obtained with job placement procedure*

We observe that, although priority was given to jobs with more urgent delivery dates, the resolution method was not able to respect the job A delivery date. In the next section, however, we will see that a schedule which respects delivery dates does exist.

If the operator is not satisfied with the scheduling result, he can manually modify it by operation moves or permutations while still respecting production routes and resource constraints. Schedule quality will then depend on the operator's experience and skills as well as his capacity to control the combinatorial aspect and various human or technical constraints which cannot be formalized.

6.2.5. *Operation placement procedure*

As we have often mentioned, schedule quality obtained with the previous procedure largely depends on the sequence in which the jobs are placed. As much as this procedure can be justified in the case of flow shops, it can become penalizing in

job shops or open shops. In fact, it runs the risk of very quickly deteriorating schedule quality for jobs which are not first in the classification.

A more generic operation placement procedure consists of no longer placing all operations corresponding to a given job before placing the operations of the following job. This procedure consists of placing operations in a certain sequence one at a time. In order to do this:

– we classify all operations corresponding to jobs to be executed in a certain order. This order or sequence takes into account certain characteristics of these operations (place in the production route criticality of the machine involved, duration, margin, etc.);

– we take the first operation in the list;

– we place the corresponding operation on the machine on which it must be executed by taking production route constraints into consideration. This placement is definitive and will not be questioned any longer in the rest of the scheduling process;

– we then place the next operation in the list and so on until it is finished.

This procedure can be considered as representing a group of progressive resolution methods for scheduling problems. In fact, as its previous description indicates, it does not explain how operations are classified: is this classification carried out once for all operations at the beginning of the resolution (and in this case, how is it done?) or is it questioned after each operation placement?

This method belongs to the class of so-called “greedy” methods within which we never challenge decisions made at any of the previous steps in the resolution. There is no *backtracking* in the arborescent resolution. Heuristics must then be chosen to make the best possible local decisions without being sure that they will lead to the optimal solution.

The simulation approach presented in the third part belongs to this class of methods and will implicitly use a systematic operation classification procedure taking increasing achievement dates into consideration.

6.3. Simulation approach

6.3.1. *Discrete event models*

Introduction

First, we will quickly present the modeling context and method of execution for discrete event simulations (for a more detailed presentation, see [CER 88] or [LAW 06]), and in the next section we will show how it can be used to solve the scheduling problem.

In this section, we are attempting to model the dynamic behavior of a production system, or more specifically, job flow circulation in this system. To illustrate the concepts and formalisms used, we will use the very simple example of a machine *M* with stock *S* waiting for execution. This machine receives jobs *i* on which it must execute operations (each job has only one operation on machine *M*) lasting a certain amount of time. Once the operation is completed, the job leaves the system.

Objects and attributes

In this type of model, we consider that the model is made up of objects (real or fictitious). In the previous example, objects are: machine *M*, stock *S* and jobs *i*. Each object is characterized by a set of attributes (small database connected to each object). Some of the attributes of these objects are fixed. These represent the characteristics of the object involved or relations with other objects. In this example, the fixed object attributes are:

– machine: *name*;

– stock: *name*, possibly *stock capacity*;

– job: *name*, *execution time on machine M.*

Other attributes are variable and evolve over time. In our example, variable attributes are:

– machine: *state of occupation* valued at 0 if the machine is idle and 1 if the machine is busy;

– stock: *list of pending jobs*;

– job: *position* in the system (“out of the system”, “in stock *S*”, “on machine *M*”).

States and events

The state of an object at a given moment is characterized by the value of all its attributes (fixed and variable). The state of the system is characterized by the state of all objects that it contains. This model is based on the notion of state. The characteristic of discrete event models comes from the fact that we are placed at such a modeling granularity that:

– the state of each component can only take discrete values. That is the case here because:

- the *state of occupation* of machines can only take value 0 (idle machine) or 1 (busy machine),

- the *state* of stock corresponds to different compositions of the list of present jobs,

- the *position* of jobs can only have three values ("out of the system", "in stock", or "on machine");

– in correlation, state changes can only happen at specific moments in time that we call "events".

In the example, there are three types of events:

– *arrival of a job* in the system. This is the moment when the job arrives from outside;

– *start of operation for a job* on the machine. This is the moment where the job arrives on the machine for execution;

– *end of operation for a job* on the machine. This is the moment when the job leaves the machine (operation completed) and exits the system.

These events correspond to the start and end of job operations on machines.

State change logic

To describe the dynamic behavior of the system, "state change logic" must be explained. State change logic describes the way in which state changes are made for the different objects involved in the occurrence of a given type of event. There is no imposed formalism to describe this state change logic. It can be described by any organization chart for representing what must be carried out during the event's occurrence.

In this state change logic, we describe:

– the different state changes which happen during the occurrence of the event;

– projected event dates for which the future occurrence is triggered by the present event's occurrence (generation of events). For example, during a "start of operation" event, we can forecast the "end of operation" event date taking into account the execution time value.

Modeling helps us consider the fact that the way in which state changes and event generation happen can depend on the state of the system during the occurrence of the event. We can then describe more complex state change logic. In the simple example used as presentation support, we can describe state changes associated with the three types of events:

1. Arrival of job i event

The change of state depends on the state of the machine at the moment where this event arrives:

– if the machine is free then:

- there is projection of the occurrence of a *start of operation* event at the current date (if the job's transfer time is negligible);

– if the machine is busy then:

- job *position* becomes: "in stock";

- the *list of jobs* in stock is incremented.

2. Start of operation of job i event

Job *position* is "on machine".

The machine goes into "busy" *state*.

There is projection of the occurrence of an *end of operation* event for job *i* at a date equal to the current date increased by execution time p_i of job *i.*

3. End of operation of job i event

Job *i* exits the system: job *position* is "out of the system".

If stock is empty then machine goes into "idle" *state.*

Otherwise, if the *job list* in stock is not empty then:

– the next job to be executed must be chosen in the *job list*;

– the name of this job must be taken out of the *job list*;

– the *position* of this job becomes: "on machine";

– there is projection of the occurrence of a *start of operation* event for the job at the current date (if the job's transfer time on the machine is negligible).

Priority rules

Choosing the next job in stock is a decision which will define the sequence in which the different jobs in stock will be processed. This decision must be made each time the machine becomes available and consists of only making a choice from jobs present in stock at that moment.

To make this decision, we can use a priority rule by allocating what we call a "priority index" to each job. The higher the index value, the higher the priority for this job and the quicker it will be processed. The fourth part of this chapter presents a classification of the most common priority rules.

6.3.2. *Discrete event simulation*

Discrete event simulation consists of reproducing the evolution of the system's state throughout time over a given timeframe with the help of the previous model. We move over time in an increasing way, emphasizing the different consecutive events: we distinguish exogenous events with outside influence (for example, the arrival of jobs) from endogenous events (consequences of state changes in the system). We use the term "simulation" because in this way, we imitate the way in which operations occur in a real system.

The most traditional resolution motor in this domain consists of increasing time by event. In fact, by definition of event modeling, the system can only change states when events occur. At each moment, we maintain a list of expected events with their time of occurrence. This list is called an "event list" (a string of chronologically ordered events). Each resolution phase consists of searching for the event with the shortest projected occurrence date. We know that there will not be a change of state between the current date and this date. This event is then the "next event" that will happen in the system. The simulation motor searches for this date and increases current simulation time to it. We then trigger the state change logic associated with this type of event. This trigger will modify the system state and possibly generate the occurrence date projection of certain events. Projected event dates are put in the event list. When we have finished processing an event, we again search for the event with the earliest occurrence date and we increase the current time to this date, and so

on and so forth until a set limit time is reached corresponding to the end of the simulation, or until the time when the schedule becomes empty.

Another type of simulation motor consists of increasing time by small increments and, testing at each increment, if conditions are met so that one of the events can occur. To categorize the occurrence of the different events sometimes a lower value time increment must be used which can be penalizing in calculation time because at each step in time, we must test if trigger conditions for each event are met.

When the simulation is finished, we have sequentially resolved the problem of determining dates for different events. We then know the state change times taking into account the state change logic associated with the different events, and in particular priority rules which can have been used to make the choices between jobs in progress.

EXAMPLE.– This example comes from the example in section 6.2. We only consider machine M2 and the three jobs A, B and C and corresponding operations. Arrival dates and execution times are given in Table 6.1.

Jobs	Availability dates	Operation time
A	5	3
B	4	5
C	0	5

Table 6.1. *Arrival dates and operation times*

Date	Type of event	Job position	Machine occupation state	Job list in stock
0	initialization		idle	empty
0	arrival C		idle	empty
0	start operation C	C --> M2	busy	empty
4	arrival B	B --> stock	busy	B
5	arrival A	A --> stock	busy	B, A
5	end operation C	C --> out	busy	B, A
5	start operation B	B --> M2	busy	A
10	end operation B	B --> out	busy	A
10	start operation A	A --> M2	busy	empty
13	end operation A	A --> out	idle	empty

Table 6.2. *Procedure of a discrete event simulation*

Completing this simulation means gradually filling Table 6.2 by using the event list. Each line corresponds to an event and reproduces the state obtained after processing the occurrence of this event.

To gradually fill this table, we have used the following schedules containing projected events at a given moment (only major event lists are presented).

First event list:

Date	Type of event
0	arrival C
4	arrival B
5	arrival A

Third event list after processing the start of operation C event:

Date	Type of event
4	arrival B
5	arrival A
5	end of operation C

Fourth event list after arrival of B:

Date	Type of event
5	arrival A
5	end of operation C

Fifth event list after arrival of A:

Date	Type of event
5	end of operation C

Seventh event list after end of operation C and start of operation B:

Date	Type of event
10	end of operation B

Ninth event list after end of operation B and start of operation A:

Date	Type of event
13	end of operation A

Throughout the simulation, we evaluate different indicators relative to the state evolution of the system over time. This is how we can:

– catalog times of certain events which have a value of importance such as job due dates. In this way, we can compare due dates to delivery dates and measure the delay resulting from priority rules used;

– measure resource utilization indices (agents, production methods, transport systems, etc.) and know their saturation state;

– measure occupation indices from different storage devices and obtain indications on their future over or undersizing;

– determine the possible occurrence of generally undesirable events (inventory location saturation, congestion, etc.). Generally, discrete event simulations highlight bottlenecks which limit the circulation of entities in the system.

Simulation is therefore a method of evaluating performance (temporal or not) of a given system considering the way the flow of jobs within a system is accomplished (priority rules, for example).

6.4. Using the simulation approach for the resolution of a scheduling problem

6.4.1. *Determination of projected schedule*

In this problem, we wish to determine completion dates of different operations $o(i, j)$ for jobs i on given resources R_k, considering temporal (allocated time, sequencing because of production routes, availability dates) as well as resource utilization constraints. We consider that jobs to be completed consist of manufacturing parts in a shop containing machines according to an order book. We thus know a group of jobs i to be manufactured on machines M_k according to their production routes.

When this shop's operation simulation has been completed, we then observe as we have seen in the previous section that we have the "film" or the "chronogram" of the history of the variations of machine states over time. We then have dates for all events and among others, start and due dates for the different operations executed on machines M_k for the different jobs. In particular, we have the completion dates which are due dates of the last operations of production routes for each job. This series of dates can then be considered projected scheduling.

This scheduling was obtained by using a method for choosing jobs in stock on machines (priority rules). No criteria can be considered in a direct way in this

choice; however, indirect consideration can be made at priority rule level (with the limitations mentioned in the fourth part of this chapter).

EXAMPLE.– Using the same example as in section 6.2.4 corresponding to a shop with three machines and three jobs to complete. A simulation using the "first in, first out" priority rule makes it possible to obtain the schedule illustrated in Figure 6.2.

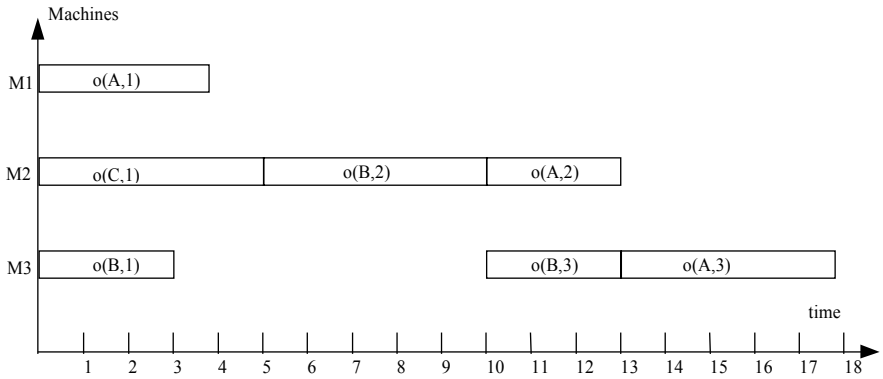


Figure 6.2. *Schedule obtained using "first in, first out" priority rule*

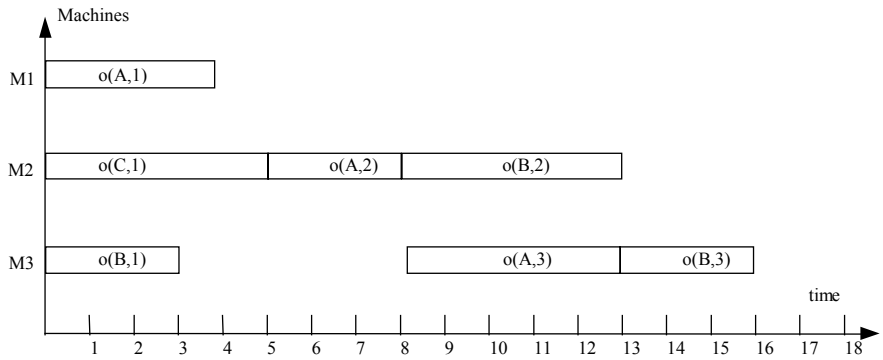


Figure 6.3. *Schedule obtained using "shortest processing time first" rule*

If we complete the same type of simulation using the "shortest processing time first" rule, we will need to execute A2 before B2 when operation C1 on M2 ends and we obtain a schedule in Figure 6.3 which respects delays.

6.4.2. *Dynamic scheduling*

Dynamic or "real-time" scheduling happens when scheduling decisions for operations on a machine are taken individually each time a machine is freed or when an operation becomes available. We can then take into consideration production contingencies which have resulted in the fact that the machine became free at a given moment. There is no pre-prepared plan any longer, but a series of decisions taken at the moment when an operation to be executed on an idle machine must be chosen. We talk about dynamic scheduling because scheduling is completed step-by-step over time.

We talk about "real-time" scheduling because the scheduling decision must be taken on site when a machine becomes physically available (or in a slightly anticipated way when we know it will become available). This decision will then be taken based on the actual state of the shop at the moment the decision is taken. It will be able to take into consideration all operations actually available for the machine at that moment (jobs present in real or virtual stock on the machine), considering disruptions which may have occurred. To make this choice, we can use different procedures. Among these procedures we can use those which are purely local and which will consist of using priority rules (see following sections). Chapter 12 presents a more elaborate procedure explaining the available degrees of freedom linked to the decision procedure and characterizing eligible scheduling classifications.

6.4.3. *Using simulation for decision support*

The availability of a simulator makes it possible to remedy the short-sightedness of priority rules not only in terms of criteria but also in terms of feasibility: respect of delivery dates, respect of maximum interval between two job constraints (constraint mainly encountered in chemical processes, particularly in surface treatment which will be discussed in Chapter 8). On the other hand, this simulation uses priority rules which are obviously short-sighted. It can only:

– provide an upper bound for the criteria to optimize;

– give proof of feasibility and not of non-feasibility.

In this way, we can use the possibilities offered by the simulation in different ways, as will be presented below.

Choice of the best priority rule

We have seen that a simulation gives information on the quality of the performance obtained by implementing a certain priority rule. To implement this simulation, we have also seen that it is important to have a list of jobs to execute with their availability and delivery dates. The performance results for the rules used are only valid for a given range of commands. If this range can be considered as representative of what the job shop will have to process, we can consider that the results are generally valid for the job shop in this environment.

We can then consider using in real time the priority rule that has turned out to be the best when used in simulation. This rule will be the one implemented in the job shop and will be used in real time to choose jobs in machine inventory.

The advantages of such as method are:

– the implementation is relatively easy and the speed of the evaluation makes it possible to test a large number of rules;

– it is reactive and makes it possible to consider the actual job shop state. We only consider the jobs actually present. Contrary to the projected approach, this approach makes it possible to react to unexpected events: a job delay will not block the completion of operations over other jobs which have not been delayed.

On the other hand, the main disadvantage comes from the fact that performance is not absolutely guaranteed. In fact, simulation has been able to evaluate performances for the range of orders used. If reality is different, we cannot extrapolate the results obtained with certainty.

Real-time decision support

The simulation can also be used to optimize certain particularly important decisions, for example because they can lead to blocking (in case of limited capacity storage) or impossibility (maximum delay between two jobs) situations. If there are only a few alternatives, they can all be simulated and the best one can be retained. If we now have to determine a chain of events instead of a single decision, then we can use optimization procedures such as, for example, simulated annealing, which uses a performance evaluation of each chain of decisions to gradually converge toward the best chain. The simulation can be used at each iteration to evaluate the performance criteria.

To develop such decision policies or to use them in the context of projected scheduling, they must be integrated in a more global job shop simulator. In this simulator, at each decision point, a certain number of anticipation type simulations must be carried out using the same model but with simplified priority rules. In this

case, the use of object languages is very appropriate, duplication and object destruction mechanisms are particularly convenient in using the same objects for all simulations and simplify all problems connected to the initialization of simulations as well as to the management of the different schedules [CAV 97].

6.5. Priority rules

6.5.1. *Introduction*

The problem of determining powerful priority rules has produced and is still producing an abundance of literature. A very large number of rules or combination of rules have been defined and tested. The analysis of this literature shows that, except for a few well known general results, it is extremely tricky to attempt to define the field of application (type of job shop, evaluation criteria) of a given rule. When analyses become more finite, conclusions from the different authors can be different, and even contradictory.

We will only describe in this part the most widely known and used rules with the different types of classifications encountered in literature and the main results for their domain of application.

6.5.2. *Description of priority rules*

Classification

We observe several classification criteria [BLA 82]:

– local or global rule: a rule is said to be local if it only takes into consideration information local to the queue, in contrast with, for example, WINQ which takes into account the load on the next machine;

Notation	Priority calculation	Description
FIFO	r_{iJ}	First in, first out
RANDOM	random draw	Random value allocated to an arrival in the queue
SPT	p_{iJ}	Shortest processing time on the machine
SPT/FIFO(α)	p_{iJ} if t- $r_{iJ} < \alpha$ 0 otherwise (FIFO)	SPT as long as wait time is lower than α, then transition to a priority queue managed by FIFO
SPT/FIFO(α)	$\min\left(p_{iJ} + \alpha, \frac{s_i}{n_i - J + 1}\right)$	Mix between SPT and SLACK/OPN (rule defined below) weighted by α
LWKR	$\sum_{j=J}^{n_i} p_{ij}$	Shortest job remaining until end of range
EDD	d_i	Earliest delivery date
SLACK	$s_i = d_i - t - \sum_{j=J}^{n_i} p_{ij}$	Smallest margin, i.e. earliest potential time until delivery date
SLACK/OPN	$\frac{s_i}{n_i - J + 1}$	Relation of the margin with the number of remaining number of operations
WINQ	$W_{i(J+1)}$	Next operation on the machine with the least jobs in progress
CRITICAL RATIO	$\frac{d_i - t}{\sum_{j=J}^{n_i} p_{ij}}$	Relation of remaining time until delivery date with the sum of remaining operation times
CoverT	$\frac{\left(\alpha \sum_{j\ J}^{n_i} w_j \quad s_i^+\right)^+}{p_{iJ} \alpha \sum_{j\ J}^{g_i} w_j}$	*Cost over Time*: relation of estimated delay cost with operation time
RR	$\left(s_i / \sum_{j=J+1}^{n_i} p_{ij}\right) e^{-c(t)} p_{iJ}$ $+ e^{c(t)} + w_{J+1}$	Weight of execution times and margin based on c(t), load of machine involved at instant t

Table 6.3. *Main priority rules*

– static or dynamic: a rule is said to be static if the priority value remains stable during the time the job remains in the queue (SPT, FIFO, etc.). A dynamic rule can retain a relative sequence between two jobs (SLACK), but the priority value must be updated again as soon as a new job arrives in the queue;

– information taken into account by the rule: rules which consider execution times, delivery dates, both, or neither, etc.;

– complexity: weighted combination of rules, usage of parameters to be set, job shop specification consideration, etc.

These classifications can be interesting in highlighting the implementation problems with rules in a simulator or directly in the shop.

Main priority rules (Table 6.3)

In addition to general job notations, we introduce the following notations:

i job involved in the operation used for priority index calculation

J rank of operation involved in job i

t current time

r_{ij} date of operation *(i, j)* arrival in job queue

S_i current job i margin (determination defined in description of SLACK rule)

w_j projected wait time for the j^{th} operation of job i involved

W_{ij} total job wait time in machine where the j^{th} operation of job i must occur

α parameter to be set by the operator

$$x^+ = \mathrm{x} \quad \text{if } x \geq 0$$
$$x^+ = 0 \quad \text{if } x < 0$$

"User priority" should also be mentioned where the user allocates a priority to each job. A certain number of priority levels are defined, and the choice between identical priority levels is made according to one of the previous rules.

6.5.3. *Experimentation conditions*

We will see that comparing performance of the different rules is particularly tricky; it is thus vital to define the simulation context in which comparisons are made rigorously. There is consensus between authors on the following conditions.

Model of shop used

A machine only executes one operation at a time.

Once an operation has started, it cannot be interrupted.

Machines are the only limited capacity resource considered.

There is no possibility for alternative routing.

Jobs are independent of one another (we do not consider assembly problems).

The priority rule is the same for all machines.

Preparation time is included in execution time and does not depend on the sequence of jobs on the machine.

Shop parameters

To be able to execute a large number of trials, the shops are randomly generated around average values defined by the user. An important problem is to determine the discriminating ones in relation to the performance of priority rules. The authors [BOU 91, GRA 93] consider that the most discriminating are dispersion of execution times, the starting margin in relation to delivery dates and shop load (average machine load rate). On the other hand, the job arrival rule (most often chosen exponentially), shop size and routing mode (randomly chosen in general) have very little influence on results [ELV 74].

Evaluation criteria

The two most often used comparison points are work-in-process and respect of delays. Shop productivity is rarely taken into account.

Work-in-process is mostly characterized by the average time jobs remain present (Favg), and variation of this time.

Respect of delays is characterized by:

– average lateness (Lavg). It should be noted that the only difference between this criteria and Favg is a constant and is therefore its equivalent in terms of optimization. It is not widely used from a practical standpoint because manufacturers are rarely concerned with early production. On the other hand, the absolute lateness value is interesting in a just in time production context;

– average real (Tavg) or maximum (Tmax) delay;

– the number of late jobs.

It should be noted that the determination of the mode of delay to be respected (delivery dates) discussed below has a direct influence over the criteria and explains that rules not considering delivery dates can be powerful in respecting these delays.

Determination modes for delivery dates

Availability and delivery dates are determined in companies by production management software (most often using the MRP approach) by evaluating the total time of transition for each job (lead time) and by assessing wait times at each workstation. The experimentations follow the same principle for realism purposes; we distinguish several modes of determination:

– CON: the allocated transition time is constant, regardless of the job;

– RAN: the allocated time is random: extreme case where the customer gets to make decisions;

– SLK: time allocated is equal to the sum of execution times plus a global wait time similar for all jobs;

– TWK: time allocated is a multiple of the sum of execution times.

Each determination mode depends on a single parameter used to set the "flexibility" in delivery dates, this parameter is strategically important for companies because it conditions the work-in-process level and market response time. A certain number of studies were carried out on the relevance of these modes ([BAK 84] for example). The goal is to minimize the average job transition time while respecting delivery dates. SLK and TWK seem more powerful, TWK often seems to be the best. This last element is surprising insofar as it would seem that the global wait time depends more on the number of waits than on execution time.

A study [CAN 88] proposing to iterate on the starting margin (delivery date minus availability date) with the help of simulations should also be mentioned. These simulations can predict real job transition times in the shop. In this way, we can separate respect of delivery dates from the optimization of work-in-process taken into consideration by a specialized priority rule.

Implementation of the simulation

In most cases, we must calculate the average and the variation of certain values. In order for the results to be significant, we must wait for the shop to have reached its permanent operation speed before starting to collect values, on the other hand, we must wait long enough for the results to be trustworthy. This last element is carried out using the evaluation of a confidence interval for the measures taken [LAW 84].

6.5.4. *Main results*

Using a single rule

For the average processing time in the shop, the SPT rule (obviously more important as distribution of execution times is high) is better in all cases. On the other hand, it is bad for the variation of this same criteria because of wait times in long operations which can be very important. Several authors have tried to resolve this flaw by mixing SPT with another rule, either alternatively or by truncation, which is more efficient. In this case, a parameter makes it possible to define from what wait time period an operation will be managed by another priority rule.

Concerning respect for delivery dates, SPT remains powerful for average real delay, particularly when delivery dates are relatively tight (margin at the start lower by five times the sum of execution times), and for the number of late jobs. However, rules explicitly taking into account delivery dates are much better for the variation of real delay. The SLACK/OPN rule is globally considered as one of the most powerful for all criteria relative to respect of delivery dates. EDD is also interesting (and simpler to implement).

The CoverT rule constitutes a particular case in the sense that it can be very powerful for real delay but it depends on the assessment quality of machine wait times. Several assessment methods for these times have been proposed: prior simulation with FIFO rule, distribution of the beginning margin on each operation, dynamic assessment in the shop. [RUS 86] studies these methods with different parameter α values and compares them to the most powerful traditional rules. For each criteria, CoverT is as powerful as the best traditional rule for the considered criteria, even with rudimentary assessments of wait times, and performance improves with the precision of these assessments.

The RR rule [RAG 93] also considers wait time assessment. It is very complete; as powerful as SPT for the average processing time in the shop, but much better for the variation, powerful for the average real delay and very powerful for maximum real delay, which is a particularly interesting criterion in the industry.

[CON 67] studied the incidence of the determination mode for delivery dates over relative performance of rules without leading to really significant results. Most of the studies take place in the TWK mode context.

Although the FIFO rule is very simple to implement, it is much less powerful than the above-mentioned rules for all traditional criteria (equivalent to RANDOM).

The "user priority" rule makes it possible to consider strategically important differences of the different jobs to be executed in the shop and to decrease the cycle time of important jobs. Using it interactively is nevertheless tricky because any priority change often has unpredictable consequences on schedule and the user quickly loses control of operations.

Finally, studies by [PIE 97] propose a different rule for each machine, and the nature of the rule is determined in real time according to the state of the shop (with thresholds with a value optimized in a finite way for the change of rules).

Mixing of rules

After focusing on the rules themselves with the use of great imagination on the part of the different authors, no significant performance improvements were reached, except perhaps in the case of the RR rule. Research is now evolving towards the study of mixing a small number of rules which are simple and complementary in terms of performance. Among the different research projects carried out, we find:

– [GRA 93] in which three rules are used: SPT, SLACK and user priority. Their use is weighted by three factors determined by a production rules-based system modeling the expertise of the relevance of rules, or by a network of neurons where information is acquired from previously treated examples. The use of fuzzy inference mechanisms enables a qualitative modeling of the expertise and eliminates threshold effects. Experiments carried out show that on the one hand, automatic learning is more powerful, but on the other hand, the rule obtained is often better than the most powerful of the three basic rules for the problem involved;

– [HOL 97] which concerns simple sums of priority indices associated with rules. The following rules have thus been examined:

- SPT+WINQ: interesting for the average processing time in the shop (the best for average and good for variation) and average for real delay,

- SPT + WINQ + FIFO, SPT + WINQ + SLACK, SPT + WINQ + FIFO + SLACK: these three rules are interesting for real maximum delay and processing time variation in the shop and weak for average values.

Generation of active schedules

In the preceding sections, only jobs actually waiting to be processed are considered in the choice of the operation to execute. In the case where execution times are very long, the decision to commit an operation can have high consequences and it might be advisable to leave a machine idle, waiting for a more interesting operation (for the performance criteria that we have set). In this way, we move away from so-called "without delay" schedules, the most often used, in which

a machine is committed as soon as a job becomes available for it. We are then placed in a more complete class of "active" schedules, which implies an intensity increase for the procedure. In fact, we must have reliable knowledge of the next arrivals, which is not always the case, and define a maximum delay for the machine without committing an available job.

Conclusion

The main conclusion is the contrast between the sum of studies completed on this subject and the lack of really indisputable results to come out and which have been discussed throughout this chapter. In particular, the attempts to constitute, notably with data analysis techniques [BOU 91], a knowledge base to help define the rule that is most adapted to a given scheduling problem (shop, production and performance criteria) have not led to sufficiently relevant expertise to be able to give a significant improvement of performance by the relevant choice of a rule in relation to the problem raised.

Automatic learning techniques based on networks of neurons [GRA 93, PIE 93] seem to be more promising as long as we stay within a relatively restricted class of shops.

6.6. Information technology tools

We must distinguish between scheduling software packages using simulation as the scheduling method and actual simulation tools.

6.6.1. *Scheduling software*

Available scheduling software packages mainly use greedy procedures which are characterized by selection modes of conflicting operations on a machine and which must be separated by a priority rule. Some of these packages use simulation to make this selection and resolve the conflicts in chronological order. They generally propose very few rules for a user to choose from (most often user priority, SLACK and SPT). Their cutting edge graphical interfaces (Gantt chart presentation, data modification, scheduling modification) and processing speed make it possible for the user to improve scheduling in an iterative way. The basic model is generally very simple (linear ranges, a single type of resource) but personalized modifications or options enable the user to consider specific operation constraints for the shop involved.

It should be noted that other selection modes are increasingly becoming available in software packages (simulation in a reactionary way from delivery dates, job placement, critical machine scheduling, etc.) making it possible for the user to choose the one which is the most appropriate for the current situation.

6.6.2. *Simulation languages*

These languages or software packages offer specific primitives for quick simulation and display of discrete event systems. Without being too exhaustive or attempting to market any one of these software packages, we can nevertheless name a few that are regularly used in laboratories or in manufacturing companies in the field of production systems: Arena [KEL 06], Witness, Modline/Qnap, Simple ++, Quest, etc. The main advantage is their ability to simulate all details of the shop to be scheduled. Although these simulators can finitely evaluate shop performance and develop a sophisticated control system, their use as scheduling systems requires graphical interface programming and interfacing with the shop environment (production management, real-time follow up), which can be cumbersome compared to scheduling software programs with this functionality already incorporated.

It should also be noted that it is possible to use spreadsheets as a tool for building simulation software dedicated to scheduling for problems of limited size.

6.7. Conclusion

The simulation approach is part of the methods called gradual schedule construction. This type of method quickly determines schedules which respects technical constraints. It is characterized by a large potential modeling power, but gradual schedule construction prohibits going back to decisions already made which could improve schedules or respect for delivery dates. In addition, schedule quality depends on its construction rules expressed in the form of priority given to the different jobs in the queue. However, we can observe that it is not always easy to adequately determine the rules for optimizing given performance criteria for schedules.

Concerning information technology tools implementing this approach, the user can choose between:

– software packages dedicated to scheduling, integrating simplified shop modeling which can be difficult to modify with a very limited number of priority rules. However, they generally make it possible to quickly develop scheduling;

– simulation packages (or spreadsheets for small problems) for detailed modeling, but with no dedicated interfaces for the scheduling problem which is mandatory when used in a shop.

6.8. Bibliography

[BAK 84] BAKER K.R., "Sequencing rules and due-dates assignment", *Management Science*, vol. 30, no. 9, p. 1093-1104, 1984.

[BLA 82] BLACKSTONE J.H., PHILLIPS D. and HOGG G.L., "A state-of-the-art survey of dispatching rules for manufacturing job-shop operations", *International Journal of Production Research*, vol. 20, no. 1, p. 27-45, 1982.

[BOU 91] BOUCON D., Ordonnancement d'atelier: aide au choix de règles de priorité, ENSAE Thesis, Toulouse, 1991.

[CAN 88] CANALS D. and CAVAILLE J.B., "Ordonnancement d'atelier par simulation: aide au lancement en fabrication par rebouclage sur les simulations", *Automatic AFCET Conference*, Grenoble, p. 191-200, 1988.

[CAV 97] CAVAILLE J.B., "Utilisation d'un langage objet pour réaliser des simulations d'anticipation", *MOSIM'97*, Rouen, 1997.

[CER 88] CERNAULT A., *La simulation des systèmes de production*, CEPADUES, Toulouse, 1988.

[CON 67] CONWAY R.W., MAXWELL W.L. and MILLER L.W., *Theory of Scheduling*, Addison Wesley, 1967.

[ELV 74] ELVERS D.A., "The sensibility of the relative effectiveness of job-shop scheduling rules with various arrival distributions", *AIIE Transactions*, vol. 6, p. 41, 1974.

[GRA 93] GRABOT B., GENESTE L. and DUPEUX A., "Ordonnancement réactif par utilisation de règles floues paramétrables", *Revue d'Automatique et de Productique Appliquées*, vol. 6, no. 3, p. 273-290, 1993.

[GRA 94] GRABOT B. and GENESTE L., "Dispatching rules in scheduling: a fuzzy approach", *International Journal of Production Research*, vol. 32, no. 4, 1994.

[HOL 97] HOLTAUS O. and RAJENDRAN C., "Efficient dispatching rules for scheduling in a job-shop", *International Journal of Production Economics*, no. 48, 1997.

[KEL 06] KELTON,W.D., *Simulation with ARENA*, 4th edition, McGraw-Hill, 2006.

[LAW 84] LAW, A.M. and KELTON, W.D., "Confidence intervals for steady state simulation: a survey of fixed sample size procedures", *Operations Research*, vol. 32, no. 6, p. 1221-1239, 1984.

[LAW 06] LAW, A.M., *Simulation Modeling and Analysis*, 4th edition, McGraw-Hill, 2006.

[PIE 93] PIERREVAL H., "Neural network to select dynamic scheduling heuristics", *Journal of Decision Systems*, vol. 2, p. 173-190, 1993.

[PIE 97] PIERREVAL H. and MEBARKI N., "Dynamic selection of dispatching rules for manufacturing system scheduling", *International Journal of Production Research*, vol. 35, no. 6, p. 1575-1591, 1997.

[RAG 93] RAGHU T.S. and RAJENDRAN C., "An efficient dynamic dispatching rules for scheduling in a job-shop", *International Journal of Production Economics,* vol. 32, p. 301-313, 1993.

[RUS 86] RUSSELL R.S., DAR-EL E. M. and TAYLOR B.W., "A comparative analysis of the CoverT job sequencing rule using various shop", *International Journal of Production Research*, vol. 27, no. 10, p. 1523-1540, 1987.

This page intentionally left blank

Chapter 7

Cyclic Production Scheduling

7.1. Introduction

The cyclic character of the rhythm of life is the most natural of things. The most logical cause is linked to natural environmental phenomena which humans have to face: the inevitable alternation of seasons due to the Earth's rotation around the Sun, as well as daily and nightly rotations due to the revolution of the Earth around its axis. Another cause is linked to traditions and society planning work based on a week containing work and rest phases. In summary, everyone lives according to a cyclic organization and this way of life naturally inspires different production management methodologies. In order to simplify and facilitate understanding of this presentation, two examples will illustrate this methodology. The first example, developed in this introduction, has no other objective than to present how activity management can be very complex and how this complexity can be sensibly reduced via cyclic organization. The second example will involve flexible manufacturing systems (FMS). It will be used as explicit support for this presentation. The introduction example involves a known, albeit complex, problem: class schedule planning within an institution (college or high school). The idea is to organize and plan an annual timetable of classes in an institution. Four types of entities gravitate around this problem: teachers, classes, classrooms and class schedules. The goal is to make sure students can attend classes corresponding to their curriculum in appropriate classrooms in the presence of a teacher specialized in the field involved.

Chapter written by Jean-Claude GENTINA, Ouajdi KORBAA and Hervé CAMUS.

Numerous capacity constraints involve the number of teachers per discipline, and the number and capacity of classrooms as well as the fact that only one teacher is assigned to teach in a given room at a given moment for students of a given class. As for the courses themselves, several characteristics must be taken into consideration. There are different course types with different timelines: classes, classroom studies (CS) and practical studies (PS). Teaching sessions are grouped into variable size modules depending on certain constraints; typically these constraints are precedence type constraints: classes before CS before PS.

The first and foremost objectives involve feasibility of *planning* over a maximum possible timeframe (school year). Once this constraint is resolved, it then becomes possible to attempt to *close the cycle as soon as possible* in order to leave as much time as possible for students before finals, all the while respecting program timing constraints for all disciplines. Another reason justifying this criterion comes from the fact that it is always important to leave the largest margin possible at the end of the year to react and adjust the timetable for unexpected events (absence of a teacher, etc.). A second criterion consists of reducing the call for substitute teachers to minimize overheads. The compromise will favor student satisfaction because the establishment is concerned about their well being and success. The image commonly associated with this type of problem involves a large table where the person responsible for the timetable works very hard inserting little cards with different colors. This job is particularly difficult because he must model in two dimensions a problem which is by definition four dimensional: classes, programs, teachers and classrooms. The solutions generally used consist of splitting this problem into sub-problems (distribution by semester for example), then for each semester, a typical week is created. This typical week will be repeated throughout this period.

The major complexity with the scheduling problem for resources (teachers and classrooms) is thus reduced to a temporal horizon; of a semester and a week respectively. This distribution of the semester load comes in fact from a first phase called short term planning. The goal of scheduling, logically called *cyclic*, thus consists of organizing the week load to be repeated over a semester as best as possible. In this chapter, we will focus on this way of organizing production activities by attempting to repeat a basic cycle relatively well optimized, thus illustrating the notion of *cyclic scheduling*. The advantage of such an approach, notably the finite optimization of this basic cycle, would remain useless without a certain control of the transition from the planning and scheduling levels. In fact, the periods retained must be carefully chosen at planning level whenever possible to represent the best compromise between the generally antagonistic feasibility of the solution retained, complexity of the approach (planning and scheduling phases) as well as optimization of previous criteria.

7.2. Cyclic scheduling problem classifications

Two classes of problems have been addressed in the context of cyclic scheduling. The first one, for reference, is the electroplating robot scheduling problem (more commonly known as hoist scheduling problem or HSP). For more detailed information, readers can refer to Chapter 8. The second application is related to an FMS job-shop which is developed in more detail here in the cyclic scheduling approach.

7.2.1. *Electroplating robot problem (HSP)*

This problem involves the organization of a flexible production mainly restricted by unique transport made up of a single capacity robot for the more simple systems. The most well known application involves electroplating chains. The basic components can be very schematically characterized by a sequence of trays in which a chemical operation with a strictly defined timeline over a time interval [min, max] must be carried out. Production sequences are different depending on the parts involved. The transport resource (made up of one or more robots on a single rail, where collision problems may occur) naturally constitutes the critical problem resource. For a given production, it is possible to search for repetitive production scheduling by the nature of products launched and their respective quantities. This comes down to repeating robot movements according to an optimized cycle [CRA 97]. It should be noted that this problem is generally NP-hard.

7.2.2. *FMS cyclic scheduling problem*

FMS represent a large class of production systems. The exact content of the term “flexibility” is subject to different interpretations. To be more precise, FMS correspond to small to medium-sized production systems with a certain number of degrees of freedom. They are halfway between mass production systems (car production, for example) where production is linear (with no routing flexibility) and “totally flexible” open shops where production is done by unit. They are relatively different from computer systems which simplify work-in-process and routings to focus mainly on operations to be executed. FMS is characterized by a set of machines (transformation resources) able to execute different types of operations (machining, tapping, turning, milling, assembly, etc.) on different types of parts around a transport system (often a type of conveyor). Different types of parts are simultaneously found in the systems and use the same machines. These parts are generally transported on a physical support (called pallet) constituting transport resources. This support is different depending on the nature of the parts class to be produced requiring specific “installations”.

There is a great analogy, without being complete, between the introduction example problem and the FMS problem that we are discussing here. In fact, *products* to be manufactured correspond to classes (in the sense of group of students); these products' *operations ranges* (commonly called jobs) correspond to class programs taken by students. In terms of problem resources, classrooms with their static constraints (for example, only physical science classes are taught in physical science classrooms) can be assimilated to *transformation resources* of FMS (such as a tooling machine, numerical command machine, etc.). It should be noted that these two types of resources represent *bottleneck* resources of the system studied. This is due to the capacity limitation of these same resources: a single classroom or a single machine for a given type of operation. A characteristic of FMS resides in routing of parts which is generally assigned to specific pallets containing loading adapted to only one type of part. In addition, the part/pallet allocation is most often invariable from the beginning to the end of production for one part. The performance criteria is to complete the requested production as soon as possible (*total production time minimization,* also called *makespan*) in order to keep some margin. It may be used to start a new production, to offset an unexpected malfunction in the production system, etc. Similarly, for our introduction example, the second objective would involve minimization of intervention from substitute teachers. Here, because of economic and production type constraints, we will *minimize the number of pallets* to be used[1] or work-in-process in the context of an invariable part/pallet allocation.

In order to facilitate understanding of the methodology, an illustrative example is developed. To simplify presentation, *Petri net* formalism is retained for the simplicity of its graphical model and also for the power of its mathematical analysis tools for the different system properties. We refer the reader to [MUR 89] and [DIC 93] for general concepts and notations commonly accepted for this formalism. This tool is first and foremost a modeling tool; it is still not used much in scheduling. Petri nets actually present the advantage of multidimensional modeling, an important advantage compared to other representation tools such as Gantt charts or other graphical models. The example considered here is taken from a project called Eowyn [KOR 98] which has enabled the comparison of different approaches (from predictive to reactive). See the flexible production cell described in Figure 7.1.

1 In general, transport resources represent supports for adapted parts very difficult to execute and thus very costly. In addition, the number of pallets in circulation leads to the increase of the number of parts and the resulting inventory value.

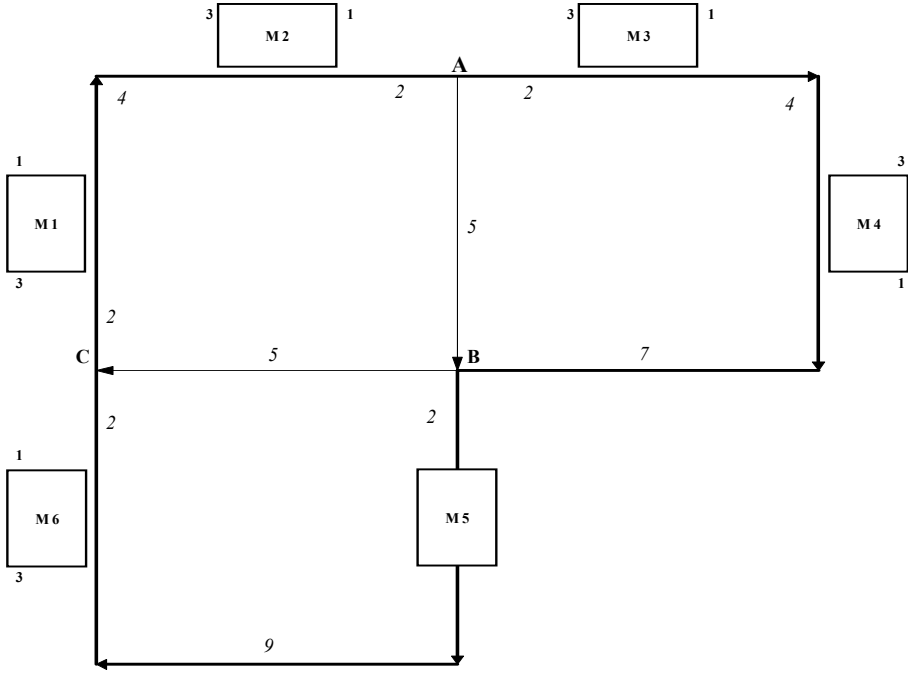


Figure 7.1. *Presentation of the flexible cell*

This cell is made up of 5 diversion stations and an on-line station (M5) around a central conveyor containing 2 switches (AB and BC represented by dotted lines). The M1 station is a loading and unloading robot enabling product arrival and exit. The other stations (M2 to M6) are multipurpose tooling machines. They can execute several different operations. In any case, these transformation resources are presumed to be non-preemptive (a part in the process of being manufactured only frees up the resource at the end of the operation) and disjunctive (each resource can only execute one operation at a time). In terms of transport system, each diversion is made up of an arrival buffer managed as a queue (first in, first out or FIFO) for 3 places, from a transformation point and a storage place at exit. The numbers indicated on the arcs represent section capacity in terms of number of pallets (for example, the section between stations M1 and M2 may contain at most 4 pallets). The complete transport system is made up of 72 cells and may contain at most 72 pallets. The pallets involved here are transport resources for circulating parts (from raw material to finished state) on adequate supports. To simplify the problem, we will assume that the pallets are dedicated (i.e. they are only able to transport parts for a single product).

We will focus on a precise production range where the parts to be produced as well as their volume are known. We consider production for three different types of parts: A, B and C. Exact quantities are also presumed given: 30 copies of each type of part. Production ratios for the different types of parts are thus easily determined (33% for each type). Determining scheduling of parts on transformation and transport resources still remains to be completed. In order to do this, we presume that operations sequences for these different types of parts are known. They are modeled with the help of Petri nets (Figure 7.2)[2]. Production of the type A part requires a sequence of 5 operations. The execution times, given and presumed set, are indicated on the ranges (for example the operation to execute on M2 for a type A part needs 5 UT[3]). There is flexibility at manufacturing range level. In fact, the third operation in this job can be equally executed on M3 or M6, but with distinct execution times. The branch containing M6 has a routing ratio (usage rate) equal to α, not yet determined. In the case of M3, it has a ratio of 1-α. Other flexibilities can be considered by the [CAM 96] method that will not be explained in detail here for purposes of clarity.

Note that range B does not carry routing ratios for the two branches that it contains. This is due to the fact that both branches are completely equal from a throughput standpoint since they use the same machines with the same execution times but with different sequences. After this quick problem presentation, it is now time to present and justify the cyclic approach.

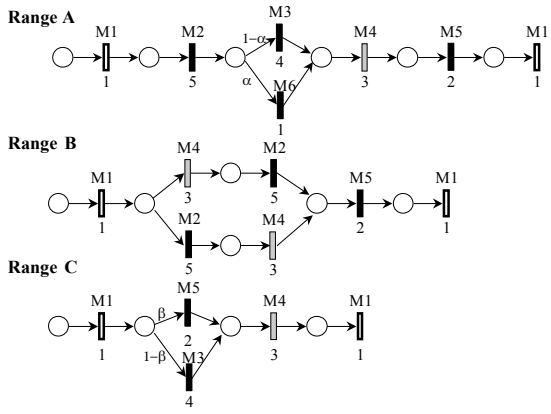


Figure 7.2. *Modeling of jobs for the different types of parts to be produced*

2 In order for the Petri net not to become too heavy, we will not present the shared resources corresponding to machines.

3 UT for unit of time. The range of this unit actually depends on the production context considered here: machining, assembly and other types of operations.

After presenting the shop floor, types of parts and expected production, the next step is *performance evaluation* consisting of determining the best branches to use for each part in order to obtain the best production flow. The goal of this step is to "linearize" jobs by choosing the best branches immediately. In addition, this step determines the optimal system flow and consequently the optimal cycle time. This cycle time is given by the bottleneck machine (with the largest workload). In fact, because work-in-process is not limited, it is always possible to use a sufficient work-in-process level to saturate the bottleneck machines to then work at its maximum speed. At this level, jobs are linearized (Figure 7.3). In this example, we presume that we will need to manufacture one part for each type of part in a production cycle. After flow optimization, the machine with the largest workload is M2 and optimal cycle time CT is equal to the load of this machine (sum of all operations that M2 produces per cycle): CT = 10 UT. This cycle time is reachable by cyclic scheduling with the use of sufficient work-in-process. The problem now consists of determining this minimal work-in-process.

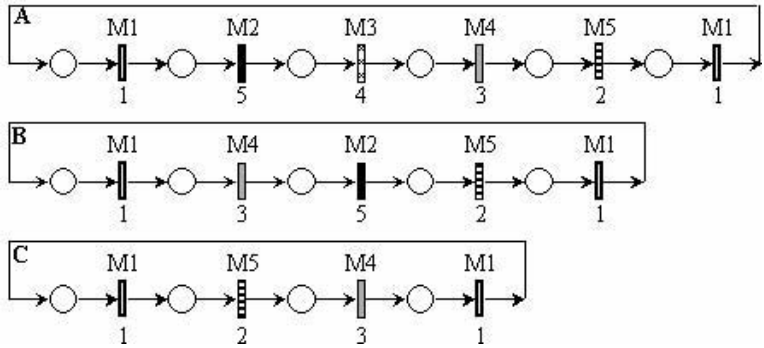


Figure 7.3. *Linear jobs after flow optimization*

7.3. Problem positioning

After the first resolution step, we have the following information: a production cycle produces three parts (1 type A, 1 B and 1 C), it lasts 10 UT and must theoretically be repeated 30 times to produce the 30 parts of each type initially ordered. System unknowns are:

– for the machines:

- the circulation sequence of parts on machines (i.e. make resource sharing deterministic),

- arrival dates for parts on machines modulo CT: i.e. the temporal allocation of transformation resources;

– for the products:

- the number of pallets used by part type,

- initial location of pallets.

During the second step (scheduling step of the permanent cyclic state), we will resolve remaining indeterminisms in order to obtain a completely deterministic cyclic system while optimizing the secondary criteria (work-in-process minimization). Scheduling is completely characterized by "*programming of task execution by allocating required resources and by setting their start times*" [GOT 93]. In our case, the idea is to allocate machines while respecting optimal cycle time (strict constraint) as well as to determine required work-in-process (performance criteria). Scheduling problems are known as being complex and highly combinatorial [BEL 82]. It was proven that project planning problems are *polynomial* combinatorial and that cyclic scheduling is *non-polynomial complete* [CAR 88, SER 89]. The consideration of transformation resources (associated with the different operations) makes the first problem *non-polynomial difficult* in most cases and leaves the second in the class of *non-polynomial* problems [SER 89]. Among the cyclic scheduling methods known to date, we should mention:

– *bottleneck machine* scheduling [ERS 82]: the main criterion for this approach is the minimization of the average and maximum size of the input buffer at each machine. It is based on bottleneck machine job scheduling (slowest machine with regards to the load) followed by conflict resolution of other machines;

– *k-cyclic* scheduling [CHR 97]: this approach focuses on cyclic scheduling in an information technology context. The work-in-process minimization objective has not been addressed (notion of work-in-process is of minimal importance in this case). In this particular case, work-in-process corresponds to memory space necessary for storing intermediate results. Since this resource is relatively cheap, its minimization is not sought after;

– *1-cyclic* scheduling [HIL 89, VAL 94, CAM 96, KOR 02]: these approaches all have the objective of respecting optimal cycle time while minimizing work-in-process.

Before presenting the scheduling algorithm minimizing work-in-process and respecting the optimal cycle time, we will show attainability for this optimal cycle time. An intuitive demonstration is the direct result of Petri net characterization. In fact, by marking all places in the linearized graph (Figure 7.3), it is clear that all bottleneck machines are saturated (as soon as a machine finishes an operation it immediately finds a part for production) and the system evolves at maximum speed, i.e. at the speed of the bottleneck machine. In this way, with the hypothesis that work-in-process is large enough, the optimal cycle time (equal to bottleneck

machine workload) is attainable. The goal of the scheduling algorithm is to determine the minimum amount of work-in-process to reach this performance (necessary and sufficient work-in-process). This approach attempts to minimize production costs (pallet production costs, intermediate inventory costs, etc.) and reduce the risk of system saturation by work-in-process that is too large and can block it. In order to evaluate the quality of results, we will calculate a lower bound of the work-in-process to be determined. We will consider job A for example: total operation duration for this job is equal to 16 UT (this is the sum of corresponding operation durations). We know that the cycle width is CT = 10 UT (cycle time). Consequently, a part can only have operations of which the sum of duration is at most equal to 10 UT at each cycle. In this way, to obtain finished part A at each cycle, *at least* two A parts are required in the system in order, for each of them, to "virtually" experience half the process. Generally, we obtain the following work-in-process lower bound for range G:

$$B(G) = \left\lceil \sum_{G \text{ operations}} \frac{\text{operation timeframe}}{\text{CT}} \right\rceil$$

where operator "$\lceil x \rceil$" represents the smallest integer higher than or equal to x. The system work-in-process lower bound is equal to the sum of lower bounds for all its jobs. This bound is not necessarily reachable.

7.4. Presentation of tools used

After an understanding of general theoretical concepts related to the cyclic scheduling problem, we will present the main modeling tools as well as notions linked to the cyclic operation. The Gantt chart will be extended in order to better represent the "flat" projection of modeled scheduling with the help of Petri nets.

7.4.1. *Modeling using Petri nets*

The interest in using Petri nets is not only limited to their implementation in the first development phases (verifying system properties such as blocking, agility and performance analysis). They can also be used judiciously throughout resolution to follow gradual transformations of the model by considering its flexibilities. Final cyclic scheduling also uses a formalism derived from Petri nets: Event Graphs (EG). These graphs are particularly well adapted to the modeling of scheduling cyclic behavior in the sense that at each end of the cycle, the system goes back to its initial state, so that it can relaunch the cycle.

Figure 7.4 represents a cyclic production by an EG: two jobs using a single machine are produced. Each job is modeled by the operation to be executed in addition to an immediate transition (serving to form the cycle). As for the machine, it is represented by a machine cycle. Since its allocation is cyclic, it is alternately allocated to one and then the other of both jobs. At the initial state (graph 1), both parts are in raw mode and the machine is ready to receive part 2. After a cycle including the production of a copy of each job, parts are finished and the machine goes back to its initial state (graph 2). The only thing to do then is to cross immediate transitions (in zero time) to find the system in its initial state to restart a new production cycle (graph 3). EGs then clearly and faithfully represent the dynamic of cyclic scheduling.

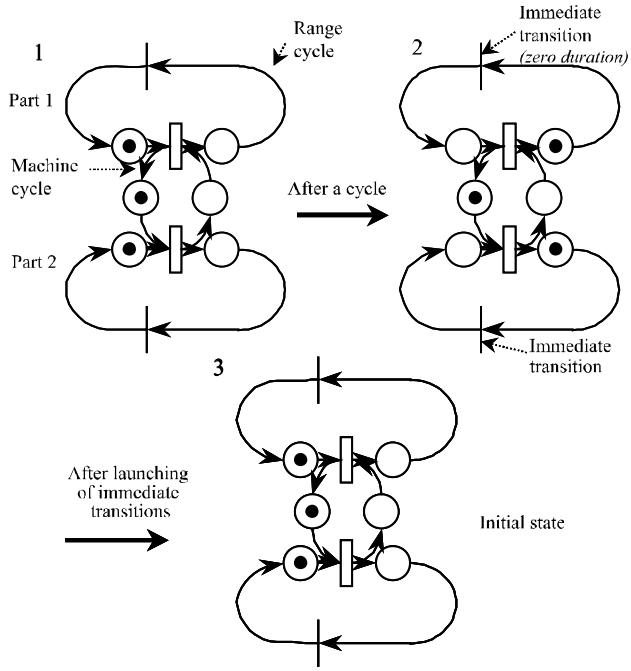


Figure 7.4. *Advantage of using an EG to model a cyclic order*

With this type of graph, it is easy to grasp the multidimensional aspect of scheduling. With the study of this system's cycles, we can observe two main characteristics: the process aspect represented by the pallets (and thus the parts allocated to pallets circulating in the production system) and the transformation resource aspect represented here by machines. The temporal aspect is associated with transition time delay characterizing the deterministic timeframe of operations.

See [HIL 89] for more information on properties of EG Petri nets. We should note that the resulting EG makes it possible to model the permanent cyclic system(s). Total production scheduling of a finite number of parts will mainly include the permanent state and transition phases [KOR 00] ensuring production start and end. The simultaneous consideration of temporary/permanent state alternation has led us to consider a sub-class of Petri nets integrating EGs: choice-free Petri nets [TER 97].

7.4.2. *Dual Gantt chart*

Although this model presents a graphical advantage because of the dynamic and multidimensional aspects of the problem, it is still difficult to mentally envisage the temporal sequence of the control because of parallelism involving products and associated resources. It can therefore be necessary to use another formalism to represent the control behavior in two dimensions, especially if we are trying to represent the resulting scheduling without referring to the explicit knowledge of Petri nets. Our proposition is to extend the Gantt chart to a dual Gantt chart. In fact, most of the time on a Gantt chart, only two dimensions are represented involving time and jobs. When scheduling is carried out gradually by consecutive placement of the different operations, as is the case with this approach, it may be interesting to simultaneously display scheduling of operations for products, at transport support level, and also for machines based on EG where we find product (job) and machine cycles in parallel. It is possible to represent the allocation of operations to corresponding machines on the Gantt chart (Figure 7.5).

In addition, it is necessary to regroup all operations for the same product. It is for this reason that we have added this new dimension in the proposed extension. The advantage of this type of representation is to enable people in charge of scheduling to understand in one chart the main scheduling parameters, as the example in Figure 7.5 shows. This example highlights the duality which exists between products and machines. In this way, considering these two aspects becomes easy for the work-in-process optimization problem. This example also illustrates the cyclic aspect of scheduling. In fact, the cycle is based on the cyclic diagram repeated for machines which imposes a periodicity and a synchronization of all production. In this example, to produce part Pr1, it is necessary to use two pallets P1 and P2. These pallets follow each other in a set order. In this way, during the first represented cycle, P1 transports part no. 1 which has just entered the system to execute the first two operations. During this time, P2 transports part no. 0, which has already been subject to these two operations, to execute the third and final operation. Consequently, during the following cycle, P1 will always transport part no. 1 for the last operation, whereas P2 will have already finished part no. 0 and will start a new cycle with part no. 2, etc.

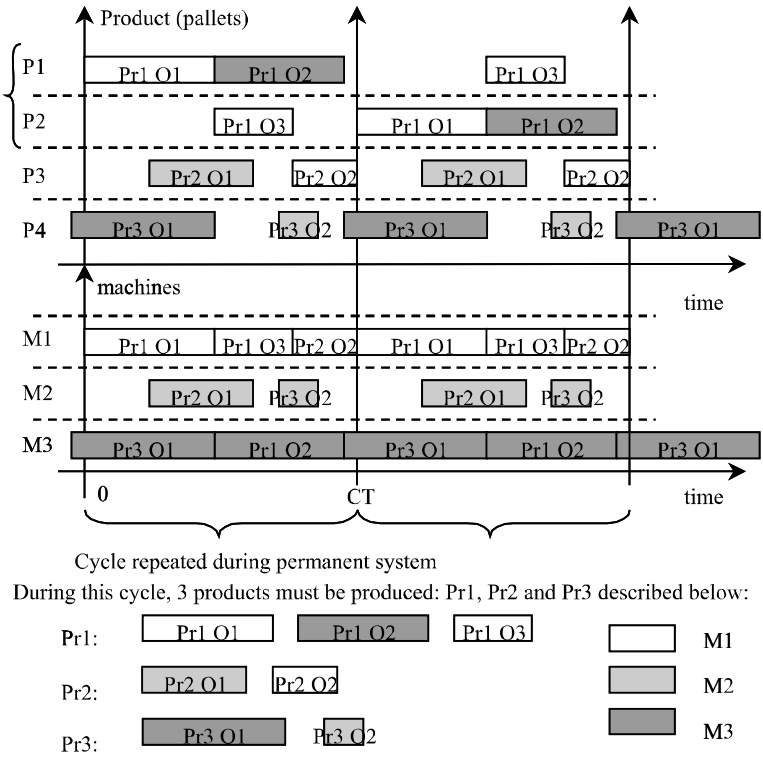


Figure 7.5. *Illustrative example of the dual Gantt chart*

7.4.3. *Resource availability interval*

We will now focus on the concept of a resource availability interval, developed in [VAL 94], to extend it to our problem. An availability interval is a time interval during which a machine is unoccupied. Knowledge of these intervals is very useful to better organize the operation placement procedure constituting the basis for the proposed scheduling method. In traditional scheduling, we generally use the notion of machine margin to evaluate their occupation ratios. It is based on the evaluation of the time periods during which the machine is idle. Cyclic scheduling is based on the same ideas (Figure 7.6).

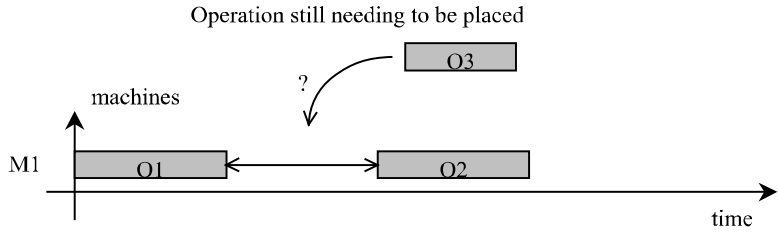


Figure 7.6. *Utilization of periods of machine inactivity*

However, it is important to define with precision the notion of a machine margin to be able to use it during scheduling. In fact, when a single cycle is fixed and an operation (O1 on Figure 7.7) is placed, two distinct periods appear during which the corresponding machine (M1) remains idle: intervals [0, t1] and [t2, CT]. Due to this, placing a new O2 type operation in one of these two intervals becomes impossible because they are both smaller in size than the operation to place. Scheduling thus seems impossible.

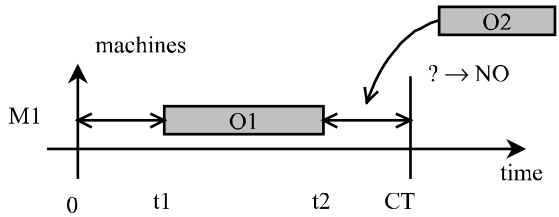


Figure 7.7. *Placement of an operation in the cycle*

However, it is possible to place this O2 operation after O1 by representing several cycles in sequence, as we have done in Figure 7.8. Both inactivity periods invoked will now only form a single availability interval in which it becomes possible to place this O2 operation. Actually, placement of O1 operation on the machine has only created a single availability interval for machine M1: interval [t2, t1 + CT], denoted as $[t2, t1]_{CT}$. The availability interval of a machine in cyclic scheduling is thus defined as the largest possible (continuous) interval *between two consecutive operations* during which the machine is idle. In this way, we define the notion of an overlapping cycle. This notion is related to cyclic scheduling for which certain operations (here O2; see Figure 7.8) "overlap" two consecutive cycles. Actually, in any cyclic scheduling, there is cycle overlapping. This overlapping obviously depends on the origin of the cycle (because of the definition of the cycle

start date). However, it is useless to be able to precisely define this cycle start. For purposes of simplicity, we refer the reader to scheduling methods which supposedly do not carry out cycle overlapping such as [HIL 89] and [VAL 94]. Such methods are most often sub-optimal and consider a set reference origin for all operations. They thus restrict availability intervals of cyclic scheduling machines and do not optimize placement of operations during scheduling.

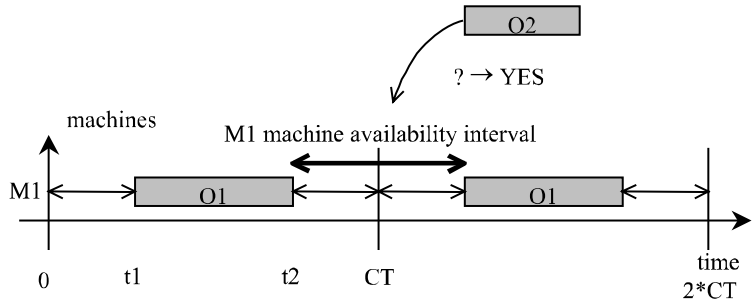


Figure 7.8. *Definition of cyclic machine availability interval*

7.4.4. *Operation placement policies in cyclic scheduling*

In order to eventually reach optimality, or at least to get as close as possible, we have had to redefine the notion of operation placement policy in scheduling. The method retained consists of a *progressive operation placement*, notably restricted in the temporal domain by the maximum CT cycle time to respect and by the number of parts of each type to produce during this cycle. In fact, the most widely used policy consists of executing the operations as soon as possible, based on a policy called the earliest firing operating mode which is no longer sufficient for reaching optimality for cyclic production. Actually this policy is only proven to be "optimal" in the case of total production time minimization and in a few specific cases [CAM 98]. We focus on minimization of the number of works-in-process in a cyclic context. Since operation placement is performed gradually (one operation at a time), the order in which operations are taken into account is therefore very important. The only constraint to respect is the precedence between operations in the same job. For the rest, the different possibilities must be studied (see section 7.5). It is necessary to constitute a sequence of operations to be scheduled (sequenced series of operations to be scheduled) before moving to the actual placement of these operations. Let us suppose that we have such a sequence and only deal with the operation placement problem (Figure 7.9).

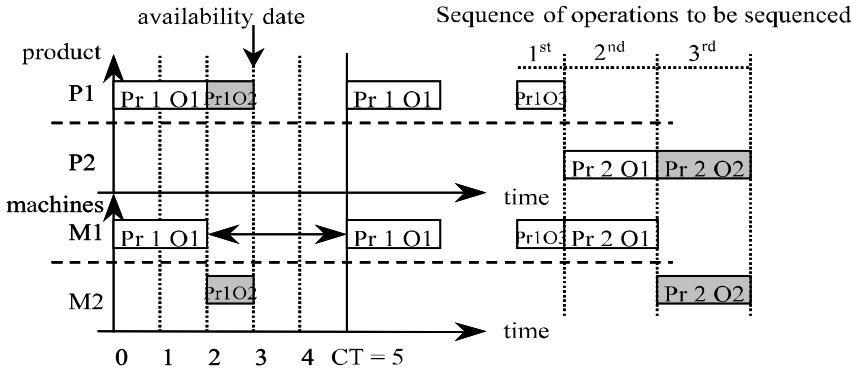


Figure 7.9. *Illustrative example of placement policies*

The cycle corresponds to the production of two parts respectively called P1 and P2. P1 is the result of three operations from Pr1 O1 to Pr1 O3 and P2 of 2 operations Pr2 O1 and Pr2 O2. We suppose that a prior study has established that the CT cycle time is equal to 5, for bottleneck machine M1. Part of the scheduling *is presumed to be done already* and it is easy to show that scheduling still remains feasible[4]. The problem consists of scheduling the last three operations. We also suppose that these operations must be scheduled in a precise order: sequence of operations to be scheduled (Figure 7.9). Several potential solutions must thus be considered for the first operation to be placed: Pr1 O3. First, we can try to place this operation as soon as possible, i.e. as soon as the product and corresponding machine are both free, in this case from date 3. This policy which attempts to minimize the number of works-in-process can lead to a deadlock, for example by not allowing placement of the next operation Pr2 O1 (Figure 7.10). This first approach tends to overly restrict scheduling by transforming initial machine availability intervals into one of smaller intervals which will not be able to be used thereafter. Since this first solution is not satisfactory, we propose other placement policies making it possible to build eligible scheduling at any time [KOR 98].

4 Each of these two machines has a time interval that is not yet allocated (availability interval) able to contain operations to schedule on the machine.

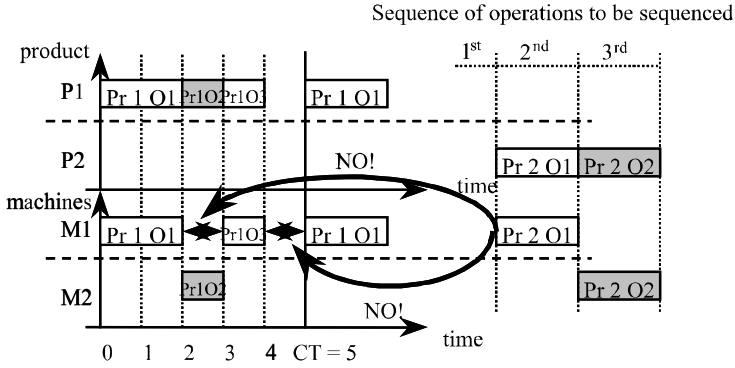


Figure 7.10. *Earliest placement policy for product*

Possible policies actually consist of placing operations in appropriate availability intervals, either at the earliest or latest in the considered interval (Figure 7.11). By applying these two policies for each operation early, we guarantee [KOR 98] that we obtain eligible and powerful scheduling. In fact, scheduling with only earliest or latest placements in the availability intervals never generates new intervals and thus does not uselessly consume machine margin. Scheduling thus found is eligible but often to the detriment of the work-in-process used because generally it does not correspond to resource availability dates, sometimes generating excessive consumption of range margin and therefore of work-in-process. That is why it is necessary to combine these different placement policies such as "earliest in the job" (to minimize work-in-process) and in the availability intervals to guarantee eligible scheduling.

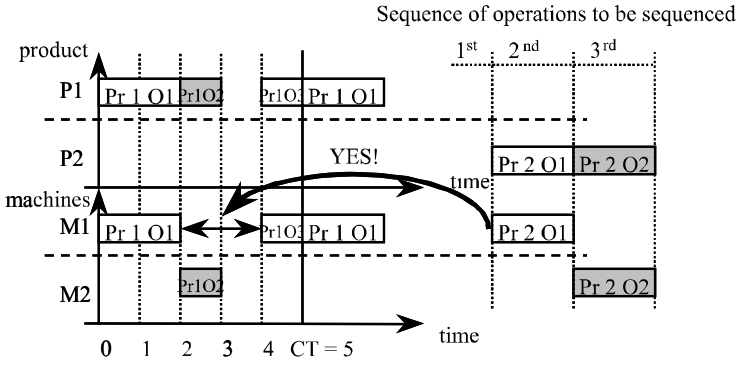


Figure 7.11. *Other placement policies: latest for availability interval*

In addition to previously mentioned variations, there are all the other operation placement policy solutions. The only real constraint involves the actual placement of an operation only within an availability interval of the corresponding machine. An optimal scheduling solution corresponds in fact to a combination of these different placement policies. Thus, in theory, to reach optimality, we must try for each operation all possible placement policies in all possible availability intervals. Because of the complexity of this exact resolution method, we have chosen to develop a heuristic using the main characteristics discussed[5]. Three placement policies will therefore be used: *earliest in the job*, *earliest in availability intervals* and *latest in availability intervals*. With these placement policies, eligible cyclic scheduling respecting the cycle time constraint equal to CT is guaranteed by minimizing the system's work-in-process as well as possible.

7.5. Algorithm principle

The algorithm proposed is based on the simultaneous use of the three scheduling policies mentioned in section 7.4. The principle is simple: for each operation, from the operation sequence to be scheduled, all three types of placement are attempted. In order to limit the combinatorial analysis, we have chosen to set the resolution based on the depth of research noted as π. This parameter corresponds to the number of operations considered at each iteration, (length of sequence of operations to be scheduled). In this way, at each iteration, the algorithm builds the depth research tree π (which corresponds to the determination of all possible sequences) and chooses the best sequence of operations from a cost function. This function is a linear function of the number of works-in-process used and machine margins.

The initialization of the algorithm is carried out by the arbitrary placement of an operation at date zero (origin of the calculating window with a width equal to the cycle time). For a better understanding of the algorithm principle, let us consider the case $\pi = 2$ with the initial example (Figure 7.3). Algorithm initialization consists of placing the first operation of job A (on machine M1) at date zero, for example[6]. At the first iteration, the algorithm will search for all possible sequences with a length of π. These sequences are restricted by precedences between the operations of a single job. In fact, strict precedence constraints between operations of a single range do not give us authority to "jump" the operations of a part hoping to place it later on. Thus, for each job where at least one operation is already placed, the future operation of the range ready for a placement is the absolute next operation in the

5 The exact method consisting of listing all possible placements is too combinatorial. Other research paths are currently being studied to develop other less greedy methods, from structural properties of the Petri net in particular.

6 For convenience reasons, it will be called A.1.

range: for our example it is the second operation of product A (using M2). In this way, still for a depth of $\pi = 2$, the fourth operation of job A cannot be present in the first sequence. On the other hand, the third operation can be part of it on the condition that the second one is present as well. This is due to the fact that the first operation of A has already been placed. Consequently, the next operation of job A to be placed can only be the second operation. However, for all other jobs still not used, all operations are candidates for the sequence. The algorithm determines all sequences. For this example, there are 68:

– 1 sequence: A.2 – A.3;

– 9 sequences: A.2 – (an operation of job B or C);

– 9 sequences: (an operation of job B or C) – A.2;

– 5 sequences: two consecutive operations of job B;

– 4 sequences: two consecutive operations of job C;

– 20 sequences: an operation of job B– an operation of job C;

– 20 sequences: an operation of job C– an operation of job B.

However, some of these sequences are equal from the point of view of their influence over the system's performance. In fact, two operations belonging to two different parts and two different machines (A.2 and B.1, for example) can be equally considered in one direction (A.2-B.1) or in another (B.1-A.2) without a difference in results. The idea is to only determine non-equal sequences, in our example, 46 of them. In fact, we only need to delete one sequence out of two in the form (operation 1 – operation 2), (operation 2 – operation 1) each time operation 1 and operation 2 belonging to two different ranges and to two different machines. Obviously, eliminating equal sequences is not so easy for $\pi > 2$.

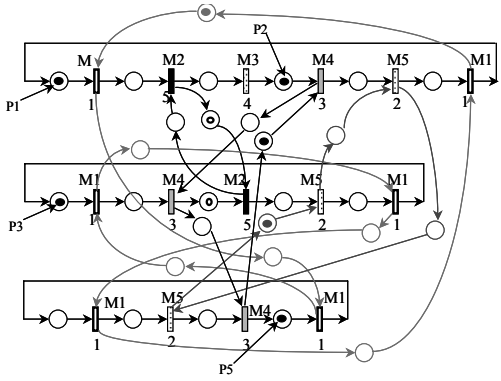


Figure 7.12. *EG modeling the permanent state*

This is how elimination is performed throughout the development of these sequences. These sequences are gradually developed with elimination as doubles appear. These sequences are then evaluated with the help of the cost function. This evaluation is carried out after virtual placement of all operations considering the three placement policies to determine the best among them. From this, we retain the first operation to actually be placed, and we restart the calculation with the remaining operations. This method is a type of controlled beam-search. The final result (the permanent state) is modeled in the form of an EG Petri net (Figure 7.12).

A dual Gantt chart can also represent such a result (Figure 7.13) where product and machine viewpoints are considered.

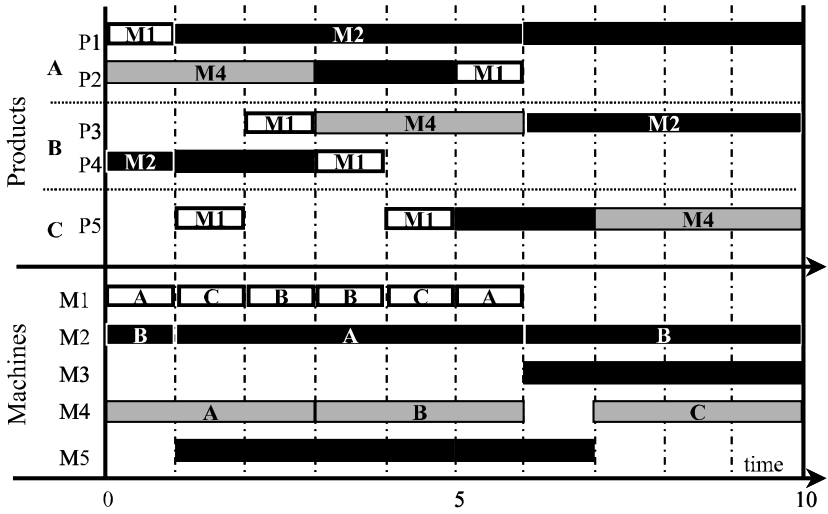


Figure 7.13. *Gantt chart representing a cycle*

In theory, this cycle is a frame to be repeated 30 times to obtain the initial request. In reality, this permanent state will last slightly less than 30 cycles because a temporary state must also be established to start production and another temporary state to end production.

7.6. Extension of cyclic strategies

During this chapter, and for purely educational reasons, we have considered an example where the initial request is “well formed”, i.e. already cyclic (30 times a

cycle of 1 part of A, 1 of B and 1 of C). We will justify this data later. In fact, production management generally expresses a global demand: 30 parts of A, 30 parts of B and 30 parts of C (Figure 7.2). It is then the turn of planning to define the ways in which to accomplish this production in cyclic function. In order to do this, we must determine the maximum theoretical flow to produce all 90 parts. This comes down to determining the optimal theoretical flow corresponding to production ratios per product: Ra = 30/90 = 1/3 of total A flow, Rb = 1/3 of B and Rc = 1/3 of C.

Note that *Zi* is the operation load of machine *Mi* brought back to a single part considered as "average"[7]. To calculate *Zi*, the idea is to find the sum, for all operations using *Mi*, of the product of execution time of the operation by the usage ratio of this operation. This ratio is calculated by the product of the production ratio of the corresponding part by the routing ratio of the branch considered as illustrated in Figure 7.2. For example, *M6* machine workload is equal to the product of the timeframe of the only operation needed (1 UT) by the production ratio of the corresponding product ($Ra = 1/3$) and by the routing ratio of the corresponding branch (α), or $Z6 = 1*\alpha*Ra = \alpha/3$.

This workload *Zi* is based on production ratios of each product: *Ra*, *Rb* and *Rc*. It is also based on routing ratios[8] α and β (ranges A and C in Figure 7.2).

We thus obtain

$$\begin{cases} Z1 = 2*Ra + 2*Rb + 2*Rc \\ Z2 = 5*Ra + 5*Rb \\ Z3 = 4*Ra*(1-\alpha) + 4*Rc*(1-\beta) \\ Z4 = 3*Ra + 3*Rb + 3*Rc \\ Z5 = 2*Ra + 2*Rb + 2*Rc*\beta \\ Z6 = 1*\alpha*Ra \end{cases}$$

or

$$\begin{cases} Z1 = 2*(Ra + Rb + Rc) \\ Z2 = 5*(Ra + Rb) \\ Z3 \le 4*(Ra + Rb) \\ Z4 = 3*(Ra + Rb + Rc) \\ Z5 = 2*(Ra + Rb + Rc*\beta) \\ Z6 < 1 \end{cases}$$

7 It is a part made up of 1/3 of A, 1/3 of B and 1/3 of C.

8 The choice of the branch for part B does not influence machine loads (*M2* and *M4*) or system flow since they both contain the same machines and production times. That is why we have not associated routing ratios with these two branches.

and

$$\begin{cases} Z1 = 2 \\ Z2 = \dfrac{5*2}{3} = \dfrac{10}{3} = 3.33 \\ Z3 \le \dfrac{4*2}{3} = \dfrac{8}{3} = 2.66 \\ Z4 = \dfrac{3*3}{3} = 3 \\ Z5 = \dfrac{2*\left(1+1+1*\beta\right)}{3} \le 2 \\ Z6 < 1 \end{cases}$$

We observe that the machine workloads are all lower than or equal to the M2 workload regardless of routing ratio values α and β. Since the optimal flow corresponds to the one on the machine with the largest load (max(Zi)), the optimal production flow cannot be higher than a part every 3.33 UT. Consequently, the theoretical lower bound of total production time is equal to 90*10/3 = 300 UT. We therefore cannot complete the initial demand in less than 300 UT. This timeframe corresponds to the continuous operation of machine M2 to execute all 90 parts. This calculation makes it possible to use the minimum production timeframe and to identify the bottleneck machine associated with this production.

The content of a production cycle still remains to be determined. Let X be the number of repetitions of the cycle and E = {n_a A, n_b B, n_c C} be the cyclic production horizon (number of parts actually produced per cycle). Although the production flow does not depend on α and β (for this example obviously), the number of parts to be produced per cycle is closely linked to these ratios. For example, if α = 1/2 (half of parts A are executed by M6 and the other half by M3), then n_a has to be in multiples of 2. In this way, the higher the α denominator (respectively β), the larger the production horizon becomes. We should add that if the production horizon is large, then the scheduling becomes more complex to determine.

For our example, the only constraints are: $n_a = n_b = n_c$ [9] and $x * n_a = 30$ [10]. An obvious solution consists of setting $n_a = n_b = n_c = 1$ and $x = 30$, which corresponds to the example considered throughout this chapter. It is easy to verify that cycle time for this example is equal to 10 and corresponds to the optimal desired flow. This situation is ideal, because the different number of parts may be first among themselves or the smallest common divider may be too large to be interesting. In fact, it would not be interesting from our point of view to retain a

9 In order to respect the initial demand (as many A as B and C).
10 In order to respect the initial A demand: the number of repetitions on the cycle multiplied by the number of A parts completed per cycle is equal to the number of A parts requested.

permanent state where the cycle would be too large (in number of parts) because it would only lead to insufficient repetition. It is thus necessary to first establish a *finite planning* phase in order to determine one or more cyclic permanent states to develop from a given initial request.

We will now use the example of an initial request for 400 parts of product A, 200 of product B and 400 of product C [CAM 98]. An obvious solution would be to produce this request by repeating a permanent cyclic system containing 5 parts (2 A, 1 B and 2 C) 200 times. However, this production can also be accomplished in two cyclic productions: the first one consists of repeating a cycle of 4 A, 3 B and 3 C 50 times and the second one consists of repeating a cycle of 4 A, 1 B and 5 C 50 times [CAM 98]. For this example, the timeframe of the solution made up of two different permanent states is less then a single state. In fact, completing the request in several cyclic systems instead of a single one can be better in terms of productivity. This results in a flexibility of production jobs. To evaluate and choose a solution, we must be able to compare the different candidate solutions. The criteria to use here are numerous: total estimated production time (at this stage in the study, we only have a first evaluation out of temporary systems and no estimate of the necessary work-in-process for these systems), solution determination complexity, number of temporary systems to calculate, etc. Several of these criteria contradict each other. This is the case notably for minimization of cycle sizes (decrease of complexity) and maximization of flow. In addition, there are numerous parameters to the problem: the number of temporary systems, the number of parts to produce by product on each cycle, the number of repetitions of each cycle, etc. [CHA 03].

7.7. Conclusion and prospects

Throughout this chapter, we have presented the main advantages of the cyclic approach to the resolution of scheduling problems. This notion seems quite natural, but this type of methodology is still rare in operation research and very rare in production. Studies by Professor Chrétienne and his team [MUN 91, HAN 94] have largely contributed to the demonstration of the richness and efficiency of such methodologies. In this case, hypotheses associated with this type of problem remain restrictive: infinite problem (or considered as such), prior in-depth knowledge of products to be manufactured throughout a cycle, etc.

The advantage of studies on cyclic scheduling in flexible manufacturing systems [HIL 89, VAL 94, CAM 97, KOR 02] resides in the fact that they address in an original manner the problem of complexity decrease for the scheduling problem with the help of production organized in repetitive production cycles. Still, in the case of cyclic scheduling, research procedures for solutions remain heuristic and cannot pretend to be optimal in theory. Generally, the combinatorial analysis of

these problems remains complex and the problem is globally NP-hard, which prevents the practical application of exact methods.

Throughout this chapter, a few paths were mentioned to almost optimally solve the cyclic scheduling problem. In fact, the progressive operation placement approach can become exact if we consider at the start scheduling of a sequence of operations with a length equal to the total number of operations to be placed in the cycle and that we propose for each operation to be scheduled all possible placement policies in the different availability intervals. This method is exact but very complex. An approach that would limit this combinatorial analysis as much as possible is in development. The most promising path seems to reside in the structural analysis of the cyclic scheduling problem taking into consideration particular properties of the Petri net model associated as well as characteristics of manufacturing production systems. It is in this spirit that we have presented this inventive method concerning cyclic scheduling. The main problem encountered is not so much in the scheduling optimization problem of a cycle as in the preliminary planning phase. It is during this phase that we search for the most appropriate cyclic scheduling strategies considering (finite) production to be completed, with criteria involving first minimization of total production time and second minimization of the system's work-in-process. In any case, the cyclic scheduling problem remains an open problem from the standpoint of resolution methods to be developed as well as from the standpoint of fields of application.

7.8. Bibliography

[BEL 82] BELLMANN R., ESOGBUE A.O. and NABESHIMA I., *Mathematical Aspects of Scheduling and Applications*, Pergamon Press, 1982.

[CAM 96] CAMUS H., OHL H., KORBAA O. and GENTINA J.-C., "Cyclic schedules in flexible manufacturing systems with flexibilities in operating sequences", *First International Workshop on Manufacturing and Petri Nets, 17th International conference on Application and Theory of Petri Nets*, p. 97-116, Japan, 1996.

[CAM 97] CAMUS H., Conduite de systèmes flexibles de production manufacturière par composition de régimes permanents cycliques: modélisation et évaluation de performances à l'aide des réseaux de Petri, Doctoral Thesis, University of Lille 1, 1997.

[CAM 98] CAMUS H., MUGARZA J.C., TERUEL E., GENTINA J.C. and SILVA M., "Scheduling Petri nets models and structural analysis", *CESA'98: IMACS Multiconference on Computational Engineering in Systems Applications*, vol. 3, p. 228-234, Tunisia, 1998.

[CAR 88] CARLIER J. and CHRETIENNE P., *Problèmes d'ordonnancement: modélisation/ complexité/algorithmes*, Masson, 1988.

[CHA 03] CHAIEB I. and KORBAA O., "Intra-cell machine layout associated to flexible production and transport systems", *Journal of Engineering Manufacture (JEM)*, 2003, vol. 217, no. 7, p. 883-897.

[CHR 97] CHRÉTIENNE P., COFFMAN E. G., LENSTRA J.K. and LIU Z., *Scheduling Theory and its Applications*, Wiley, 1997.

[CRA 97] CRAMA Y., "Combinatorial optimisation models for production scheduling in automated manufacturing systems", *European Journal of Operational Research*, 1997, no. 99, p. 136-153.

[DIC 93] DI CESARE F., HARHALAKIS G., PROTH J.M., SILVA M. and VERNADAT F.B., *Practice of Petri Nets in Manufacturing*, Chapman and Hall, 1993.

[ERS 82] ERSCHLER J., LÉVÈQUE D. and ROUBELLAT F., "Periodic loading of flexible manufacturing systems", *IFIP Congress*, APMS, p. 327-339, France, 1982.

[FRE 88] FREIN Y., DALLERY Y., PIERRA J.-J. and DAVID R., "Optimisation du routage des pièces dans un atelier flexible par des méthodes analytiques", *APII*, vol. 2, no. 5, p. 489-508, 1988.

[GOT 93] GOTHA, "Les problèmes d'ordonnancement", *RAIRO-Recherche Opérationnelle*, vol. 27, no. 1, p. 77-150, 1993.

[HAN 94] HANEN C., Problèmes d'ordonnancement cycliques, Habilitation à diriger des recherches, Blaise Pascal Institute, Paris VI University, 1995.

[HIL 89] HILLION H.P. and PROTH J.M., "Performance Evaluation of job-shop systems using timed event-graphs", *IEEE Transactions on Automatic Control*, vol. 34, no.1, p. 3-9, 1989.

[KOR 98] KORBAA O., Commande cyclique des systèmes flexibles de production manufacturière à l'aide des réseaux de Petri: de la planification à l'ordonnancement des régimes transitoires, Doctoral Thesis, University of Lille 1, 1998.

[KOR 00] KORBAA O., YIM P. and GENTINA J.-C., "Solving Transient Scheduling Problems with constraint programming", *European Journal of Control*, vol. 6, no. 6, p. 511-524, 2000.

[KOR 02] KORBAA O., CAMUS H. and GENTINA J.-C., "A New Cyclic Scheduling Algorithm for Flexible Manufacturing Systems", *International Journal of Flexible Manufacturing Systems (IJFMS)*, vol. 14, no. 2, p. 173-187, 2002.

[MUN 91] MUNIER A., "Résolution d'un problème d'ordonnancement cyclique à itérations indépendantes et contraintes de ressources", *RAIRO-Recherche Opérationnelle*, vol. 25, no. 2, p. 161-182, 1991.

[MUR 89] MURATA T., "Petri nets: properties, analysis and application", *IEEE Conference*, vol. 77, no. 4, p. 541-580, 1989.

[SER 89] SERAFINI P. and UKOVICH W., "A mathematical model for periodic scheduling problems", *SIAM Journal of Discrete Mathematics*, vol. 2, no. 4, p. 550-581, 1989.

[SIF 80] SIFAKIS J., "Performance evaluation of system using nets, net theory and applications", *Lecture Notes in Computer Science*, Springer-Verlag, p. 307-319, 1980.

[TER 97] TERUEL E., COLOM J.M. and SILVA M., "Choice-free Petri nets: a model for deterministic concurrent systems with bulk services and arrivals", *IEEE Transactions on System, Man and Cybernetics*, vol. 27, no. 1, 1997.

[VAL 94] VALENTIN C., "Modeling and analysis methods for a class of hybrid dynamic systems", *Symposium Automatisation des Processus Mixtes: Les Systèmes Dynamiques Hybrides, ADPM'94*, p. 221-226, Belgium, 1994.

This page intentionally left blank

Chapter 8

Hoist Scheduling Problem

8.1. Introduction

Since the 1970s, and more specifically since the founding works by Phillips and Unger [PHI 76], automated electroplating systems have been widely studied. Such systems are composed of tanks containing chemical or electrolytic baths in which parts (or *products*) to be processed are submerged. Transportation of products between tanks is performed by one (or more) hoist(s). When the production line is dedicated to a single product type (printed circuit boards, for example), hoist movement sequence is cyclic and an open loop control is used. Small series production lines, or even unit production lines are automated in closed loop because hoist movements are no longer cyclic. Several scheduling problems encountered in this context have aroused the interest of the scientific community. Since they all involve hoist movement scheduling, the generic name "*Hoist Scheduling Problem*" or *HSP* is generally used to refer to them in scientific literature. However, they involve different characteristics: some of these are static (or even cyclic) and others are dynamic. This chapter presents them, paying particular attention to:

– the physical system and production constraints to consider;

– a classification of scheduling problems encountered;

– the main cyclic scheduling approaches in scientific literature;

– studies dedicated to the resolution of non-periodic problems.

Chapter written by Christelle BLOCH, Marie-Ange MANIER, Pierre BAPTISTE and Christophe VARNIER.

8.2. Physical system and production constraints

An electroplating line is generally made up of (Figure 8.1):

– one or more *arrival* and *exit stations*,

– *carriers* on which parts to produce are grouped into batches,

– *tanks*, often lined up, containing chemical solutions,

– *hoists*, or *robots*, ensuring transportation of carriers between tanks.

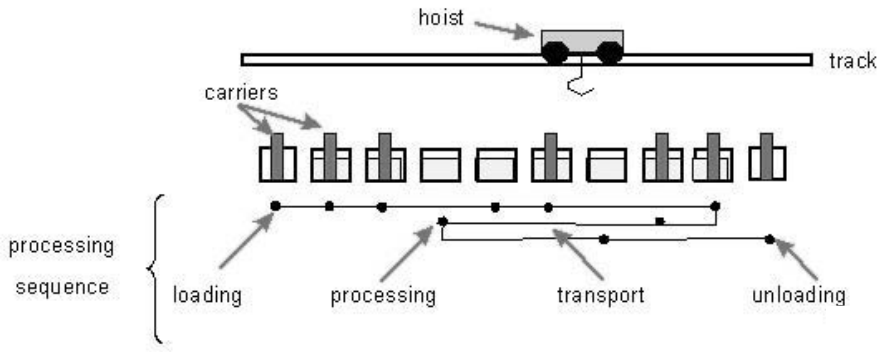


Figure 8.1. *Basic line*

The carrier is an entity which groups parts that are to be treated in an identical way. The quantity of assembled parts is a specification in the same way as *the processing sequence* associated with the carrier. This specification is an ordered set of operations that it must undergo (described by the station where each operation must be performed and the corresponding processing time). It generally contains the following phases:

– loading of parts on a carrier performed at the arrival station;

– preparation (cleaning and rinsing of parts);

– the optional pre-treatment, by applying one or more sub-layers;

– metal deposit electrolytically or chemically;

– finishing (passivation by phosphate treatment, whirling, drying, etc.);

– unloading at the exit station.

The processing specifications can also include operations executed after unloading of parts (stripping or rinsing, for example) for cleaning the carrier.

Processing operations (or *treatments*) cannot be interrupted (no *preemption* is allowed). In addition, no wait is allowed between two successive processing operations in order to avoid deterioration when parts are exposed to air. When a carrier is taken out of a tank after a processing operation, it must be transported as fast as possible into the tank where the next operation will take place. Because of this, intermediate storage between tanks is not possible. Hoists must not stop when they transport a carrier.

8.2.1. *Tanks*

Tanks contain chemical solutions into which carriers are soaked. Processing time in each bath (or *operation duration*) is limited: most often, chemical experts responsible for production define a minimum processing time (needed to obtain desired quality) and a maximum time beyond which the product:

– has not changed (total indifference);

– has tolerable deterioration (which should generally be identical on a complete series for consistency);

– becomes defective, because the processing time no longer corresponds to the chemical characteristics of the treatment bath previously set.

In a standard case, each tank receives one carrier at a time (it is then called a *single-capacity tank*) and carriers are only processed once into each tank. There are, however, several extensions to this basic model:

– if carriers are processed more than once in the same tank, it is called *multi-function* (Figure 8.2) otherwise it is called *mono-function*. This additional degree of freedom carries a blocking risk, even if the maximum time constraint is relaxed. In addition, it makes variable indexing trickier;

– if a treatment is particularly long, creating several places in a larger tank, called a *multi-capacity tank* (or *duplicated tank*) is advantageous (Figure 8.2). This complicates the problem since the distance matrix between tanks is no longer computable. The operation must be assigned to one of the places in the tank to know exact distances. Most often, this assignment is not carried out beforehand and only the greatest distance is considered, but this simplification can be tricky when multi-capacity tank places are not consecutive.

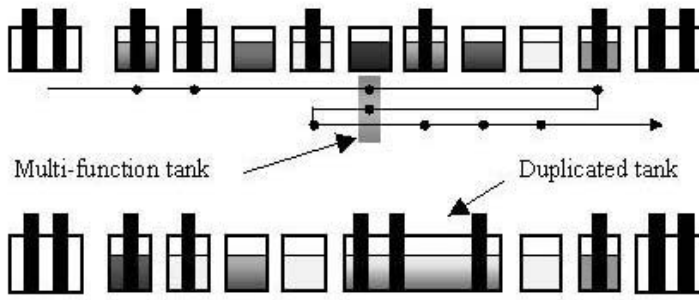


Figure 8.2. *Variation on processing resources*

8.2.2. *Hoists*

When a small production line has only one hoist it is called a *single-hoist,* otherwise it is called *a multi-hoist* (Figure 8.3). In practice, the number of hoists varies from 2 to 6, but is generally between 2 and 3. Mostly identical, they move on a rail set along the tanks, not allowing crossovers. In addition to transport operations, they may also move while empty to let another hoist carrying a carrier pass through. Transport must be assigned to hoists and collision risks appear. To avoid any interference, a transport can require the addition of clearing movements of other hoists.

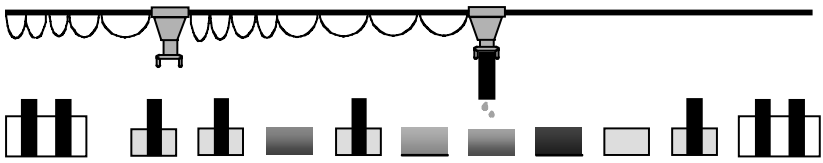


Figure 8.3. *Diagram of a line segment with two hoists*

The execution of a transport operation is complex: the empty hoist joins the tank containing the carrier, comes down, catches and lifts it. It then stops for a set draining time (to avoid contaminating neighboring baths), moves toward the destination tank, takes another pause for stabilizing (to limit carrier oscillation) and places the carrier. Movement time between two tanks can be specified by a time matrix or computed more precisely (taking into account acceleration, deceleration and nominal speed). Generally, two speeds are used depending on whether the hoist moves while empty or with a load.

Complex lines also use other types of transport systems, *transfers,* which transport carriers between several line segments (often called "*in I*" because of their linear configuration). They are generally small cars moving on rails perpendicular to the one supporting the hoists (Figure 8.4). Their number depends on line dimensions (number of I segments and number of tanks for each of them), but is often between 1 and 3. Synchronization between the transfer and a hoist is sometimes necessary for carrier exchange. The robot places the carrier in the little car transporting it up to the segment on which it must continue its treatment, and then another hoist picks it up.

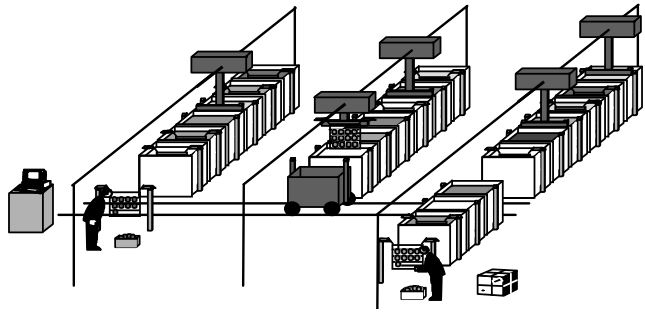


Figure 8.4. *Diagram of a complex system with a transfer*

Scheduling problems involved in this context generally concern the management of all these handling resources. The scheduling to be determined is then represented in the form of a specific chronogram, called *time-way chart*, and commonly used by line designers and users to represent hoist movements between tanks (Figure 8.5).

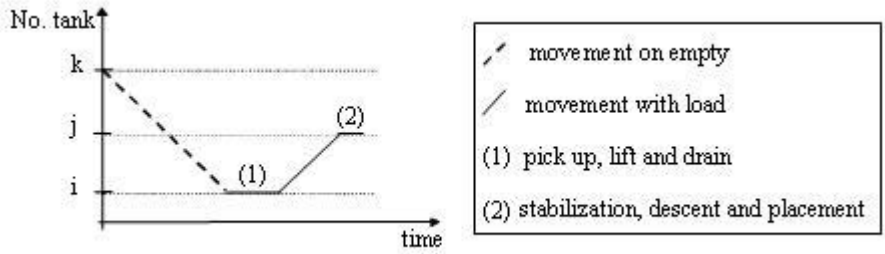


Figure 8.5. *Graphical representation of a transport operation*

8.2.3. *Carriers*

Generally, there are enough carriers and they are not considered as resources. However, sometimes, they cannot come off the line after unloading and their number is limited. They might hinder the progress of current active carriers because they are temporarily stored into unoccupied tanks. They must be moved when treatment of a full carrier requires the tank in which they are lying. This makes control even trickier. In reality, this case happens only rarely and will not be addressed in this chapter.

8.3. Hoist scheduling problems

8.3.1. *General presentation*

The common notion of a *job* used in scheduling can be associated with the carrier or parts, knowing that between loading and unloading, these entities are taken together. It seems even more natural to consider the carrier if the processing specifications include cleaning operations (see section 8.2). However, carriers may be critical resources (see section 8.2.3) and the parts arrival rule does not depend on their availability. Linking the concept of job to the carrier can turn out to be inappropriate. However, this approach is used in this chapter, because it is the one used by most authors, and cases where it is not applicable still remain relatively rare.

Electroplating includes two types of *tasks* (processing and transportation), and two types of resources (tanks and hoists). However, the conjunction of no-wait and no preemption constraints links processing operations to transport. In particular, hoist movement durations must be taken into account. The problem consists of scheduling transport in order to optimize one or more criteria, respecting:

– constraints relative to one or more product types:

  - the sequence of operations for each carrier,

  - bounded operation times, draining and stabilization times;

– no wait and no-preemption constraints, which lead to:

  - absence of intermediate buffer between tanks,

  - obligation to consider robot movement durations;

– constraints linked to resources:

  - tank capacity (single-capacity tank or multi-capacity tank),

  - number of available hoists and their capacity,

- possibly the number of carriers and the fact they must stay in the line (see section 8.2.3);

– spatial constraints leading to the risk of collision between hoists.

For each transport, starting and ending tanks and thus transport duration are known. However, the hoist's current position before the transport is performed depends on the sequence of movements. The time required to move whilst empty toward the starting tank is a variable to be determined with scheduling (such as exact processing times). In addition, an assignment sub-problem appears if the line contains multi-capacity tanks or several hoists. Assignment may be static (it precedes calculation) or dynamic (it is done during resolution).

This presentation shows that the hoist scheduling problem attracts many constraints. However, processing time margins, and especially larger ones, allow us to momentarily store jobs in the corresponding tanks as long as the maximum boundary is respected. Although this general HSP description stands for all existing cases, the scheduling problems involved often have different optimization criteria and characteristics, greatly linked to studied physical systems, to processing constraints and to considered modes of control. The approach depends particularly on the compromise to be reached between the three main objectives which are productivity, quality and flexibility. The following section provides a general overview of these different problems, by linking them to corresponding production modes.

8.3.2. *Static scheduling problems*

Hoist systems can be used in very different contexts. In very large series, consecutive jobs are all identical and production is said to be *mono-product*. The transport sequence is *cyclic*: the same sequence of hoist movements is repeated over π consecutive periods. We speak of *cycle of degree n* or *n-periodic cycle* when *n* jobs are introduced in the system throughout a period. Most manufacturers settle for 1-periodic solutions even though we know that they are not dominating solutions. This cyclic operation mode has led to the definition of a first hoist scheduling problem known as a cyclic hoist scheduling problem or CHSP [LEI 89a]. Its goal is mainly represented in terms of productivity and quality. The searched solution is the cycle maximizing line productivity, minimizing *period* π for a set cycle degree *n*. It specifies the sequence of transports, as well as the start date of each one throughout a period and must be feasible, i.e.:

– describing a sequence of movements physically executable by the hoists;

– inducing operation times compliant to specified boundaries.

The simplest CHSP, called a *basic problem* [MAN 94b], consists of searching for the minimal 1-cycle period in the case of simplest lines (i.e. single-hoist I only containing single-capacity tanks and mono-function tanks). Even though some of the constraints are ignored, Lei and Wang showed that it was NP-complete [LEI 89b]. This explains why most researchers only solved this simplest case.

In large series, the line processes different types of jobs (*multi-product* case). However, a cyclic management mode can still be considered if there are only a few types of products (two or three) and if they are quite similar. Nevertheless, the proportion of each job type to process must be known and the problem then contains an additional decision level: since jobs are not identical, building the hoist movement sequence must be preceded by (or accompanied by) a choice of product type mixing. The same job sequence periodically enters the system and the sequence repeated by hoists includes all transports required for processing all jobs involved. This cyclic problem can be considered as a multi-product CHSP.

If such a cyclic mode cannot be used, an alternative consists of working *by runs*. This management mode is a chain of mono-product production phases. The main problem then occurs during production change. The simplest solution consists of completely emptying the processing line before starting a new run. However, it is worth enabling the cohabitation of two job types in the system during the transient phase, in order to minimize its duration. This is the subject of the third type of *static* HSP.

Finally, when the line must process various parts in small series, producing by mixing or by runs is no longer possible. However, if a quite stable production plan exists, this production context can still lead to the definition of a *static* problem. If information delays are sufficient, a first decision level consists of scheduling the different jobs to process soon to achieve productivity gains and improve the quality of treatments. On the one hand, the arrival order of jobs greatly influences total production time. On the other hand, wisely choosing job arrival dates limits the number of resource conflicts between carriers and reduces the number of treatments whose duration exceeds the specified maximal processing time.

By contrast with cyclic problems commonly referenced by the CHSP acronym, the last two problems (*static but not cyclic*) presented above were grouped under the generic title of "predictive hoist scheduling problem" or PHSP [BLO 99b, ROS 99].

8.3.3. *Dynamic scheduling problems*

Sub-contracting companies must generally treat a large variety of jobs in small quantities and in very short delays. Faced with the decreasing size of series and

reduction in delays, they must combine various jobs on the line and increase the flexibility of their automated systems. The increasingly short delays imposed by their prime manufacturers do not enable them to establish reliable planning of several hours, and manufacturing managers must be able to react quickly (to an urgent order for instance). Professionals say that the line is managed in *random mode*. There are two main approaches in hoist movement scheduling based on the trade-off to reach between three main objectives (*productivity*, *quality* and *flexibility*):

– if quality of treatments is the main objective, all transports are scheduled before letting any new carrier enter the system. This forecasted scheduling must consider all present carriers (in tanks or at loading station) and respect all constraints. In particular, new jobs must be able to come in as soon as possible, while making sure (among other things) that all maximum processing times will be respected and that no job will become defective. Generally, the criterion to minimize is the *makespan* (i.e. end date for all jobs). This approach is very much decried in the manufacturing world as not being robust enough, because it suggests transport start dates which must be strictly respected, which is almost impossible in reality because of unavoidable production hazards. Nevertheless, this solution remains the only one suitable for high added value production runs for which treatments must be of very high quality (in the aerospace field, for example). The associated scheduling problem is called the "dynamic hoist scheduling problem" or DHSP [LAM 96b];

– if priority is given to *productivity* and *flexibility* (in terms of job diversity), transport tasks are dynamically assigned to hoists. A hoist control system generally relies on a list algorithm that determines which carrier will move next each time a hoist ends a transport. This decision is generally made in relation to the nature of the bath in which the carrier will stay and the time it will spend there before exceeding its maximum processing time. Respect for maximum boundaries is not always ensured, but in certain cases, it is possible to limit the negative consequences of such a violation (by decreasing the intensity circulating in the bath, if the operation involved is electrolysis, for example). This second approach is called "reactive hoist scheduling problem" or RHSP [BLO 97] and is different from the previous one by the decision horizon involved. The DHSP corresponds to "*by job*" management whereas RHSP corresponds to "*by task*".

8.3.4. *Classification and brief state of the art*

The wealth of literature dedicated to HSP combined with the diversity of cases treated makes classification essential. An extension of the normal notation used in scheduling [GRA 79] was proposed in [BLO 99b] and [MAN 03]. It enables us to situate the HSP to be solved among the various existing instances, and to identify the class to which it belongs. It considers some of the main physical and logical

parameters found in the literature related to the HSP. It covers four fields: type of HSP, physical parameters, logical parameters and criteria. Each field consists of several parameters:

– type of HSP: the HSP tackled can be static (cyclic (*CHSP*) or not cyclic (*PHSP*)), or dynamic (dynamic (*DHSP*) or reactive problems (*RHSP*);

– physical parameters: this field respectively includes the number of lines (*nl*), the number of transfers connecting these lines (*ntransfer*), the need of synchronization between hoists and transfers (*synchro*). It also provides, for each line *i* of the facility (*i*=1 to *nl*), the number of hoists (*mh*), the number of tanks (*mt*), the number of carriers (*nc*), the maximal capacity of tanks (*ct*), the constraints involved by the characteristics of carriers (circulation of products (*circ*), a dedicated transport system to ensure the return of empty carriers from the unloading station to the loading station (*ret*), empty carriers remaining on the line if there is no storage place near the facility (*empty*)), and finally the configuration of the loading and unloading stations: associated or dissociated stations (*load-unload*);

– logical parameters describe the production environment to be considered: the total number of parts to be treated (*nparts*), the number of different processing specifications (*nps*), the maximal number of operations among those processing sequences (*nop*), the possible cleaning of empty carriers after the unloading operation (*clean*) (one or several operations included in *nop*), and finally the recirculation constraint (*recrc*) involved by multi-function tanks;

– criteria: this field gathers one or several objectives to be optimized. The literature dedicated to HSP includes several criteria, for instance: minimize the cycle time for the CHSP (*Tmin*), minimize the makespan (*Cmax*), maximize the throughput (*ThroughputMax*), minimize the temporary period between two mono-product cycles (*TransMin*), minimize the number of hoists (*mhmin*), minimize the waste in environmental issues (*WasteMin*), minimize the number of defective jobs (*DefectiveJobsMin*), maximize the quality of treatments (*Qmax*). The generic value "Other" can be used if the considered criteria does not belong to the above list, and the notation can easily be extended if necessary.

To summarize, the complete notation is: ***XHSP* | *nl, ntransfer, synchro, (mh, mt, ct)*$_{i=1\ to\ nl}$*/nc, circ, ret, empty/load-unload* | *nparts/nps, nop, clean, recrc* | *criteria***.

This has enabled its authors to classify approximately 100 publications in tree form, by defining one tree per acronym identified in the previous section, giving a global view of the research carried out. An associated state of the art is detailed in [ROS 99]. Table 8.1 provides a summary and offers some references both to the main publications and to more recent works. The complete notation associated with each of these publications is provided in section 8.8.

This bibliographical work shows that most authors consider a relatively simple line (in I, single-hoist and single-capacity tanks). This naturally prevents us from applying the proposed solving approaches in real contexts (because industrial facilities are generally more complex). This does not mean that researchers are not interested in real production lines. However, the problem is so complex (see section 8.3.2) that the most practical path consists of addressing the simplest cases, and then studying possible extensions. Unfortunately, these might sometimes be impossible to implement (notably because of their too large combinatorial).

CHSP	*Single-hoist, single-capacity tanks and mono-product:* [PHI 76, LEI 93a, ARM 94, SON 95, LIM 97, CHE 98, MAT 00, SUB 06]
	Single-hoist, single-capacity tanks and multi-product: [PTU 95, VAR 00, MAT 06]
	Single -hoist, multi-capacity tanks and mono-product: [SHA 88, LEI 89a, HAN 93, LEI 94, LIU 02, ZHO 03, XU 04, KUN 06]
	Multi-hoist, single-capacity tanks and mono-product: [LEI 93b, YAN 01, MAN 06, MAN 08]
	Multi-hoist, multi-capacity tanks and mono-product: [LEI 91, HAN 94, MAN 94b, ARM 96, BAP 96, RIE 02, VAR 97, MAN 00]
PHSP	*Transient phase scheduling:* [VAR 96]
	Job scheduling at the system entry: [CAU 95, FLE 95, CAU 97, LAC 98, BLO 99a, ROS 99, FLE 01]
DHSP	*Single-hoist and single-capacity tanks:* [YIH 94, CHE 95, GE 95, ROS 99, FAR 01, HIN 04, PAU 07]
	Single-hoist and duplicated tanks: [SPA 99]
	Multi-hoist: [LAM 96a]
RHSP	*Single-hoist and single-capacity tanks:* [YIH 93]
	Multi-hoist: [THE 90, JEG 06]

Table 8.1. *Some publications relating to the different classes of HSP*

The presented summary also shows that much effort has been expended to solve the HSP, while emphasizing that this has not been equally distributed between the different existing variants of the problem. These contrasts are clearly linked to the evolution of production conditions over time. In fact, the first research studies were carried out when mass production was still dominant: only the mono-product cyclic problem was really interesting for companies. This explains why the search for an optimal cycle in this context has led to more than half of the listed publications. These studies resulted in the development of relatively efficient optimal algorithms,

making it possible to schedule larger production lines relevant to the industry. In addition, even though there are few facilities dedicated to a single type of product today, there are still some in certain sectors and these algorithms are still current.

Gradually, series size and delays have decreased. A growing need for increasing processing system flexibility has arisen. This has led to the emergence of research relative to lower production horizons: real-time task assignment (RHSP), mixing of jobs and transient phase scheduling between mono-product production phases (PHSP). RHSP was the first approach developed in the industry, obviously because available automatisms, specifically industrial logic controllers already used to control hoists, allowed it to be implemented quickly. Another reason was also probably because this type of control is the one offering the greatest flexibility (see section 8.3.3) and robustness (when production hitches occur). The success of this solution, the generalization of its use, may have impeded the development of other suitable paths and limited its improvement. This accounts for the low number of publications listed for the multi-product CHSP and optimization of transient phases, and for the near-total absence of recent research projects relative to RHSP.

However, this hoist control mode does not always respect the specified maximum processing times. This defect motivated research on arriving job scheduling and DHSP. Since these problems have only recently appeared in reaction to existing system failures, there are fewer publications about them than those related to CHSP. They have however multiplied in recent years and this interest may certainly reduce this difference over time. Transient phase scheduling has scarcely been studied, whereas mixing of product types seems to benefit from renewed interest. Some production line designers would now like to direct their research toward hybrid control, which would adapt hoist control to the nature of jobs entering the system. An open loop cyclic control would be used when jobs arriving over a given horizon are identical and dynamic control would be used when a different job type arrived.

Finally, this chapter would not be complete if while highlighting the problems, it did not address the methods used to solve them. Once more, it seems impossible to explain each one of the listed publications (which are described in [MAN 94a, LAM 96b, ROS 99, MAN 03]). Only a few of the main solving approaches and models will be presented. Two models in particular, developed for CHSP, exist on which most of the other approaches proposed thereafter are based. Section 8.4 will start with their presentation and will then describe a few CHSP resolution methods, before giving information on studies related to other HSP classes.

8.4. Modeling and resolution

8.4.1. *Notations*

A definition of notations used in this chapter is necessary in order to describe the presented models. To facilitate understanding and avoid any confusion with certain common notations used in scheduling, these notations have been reviewed and unified. They are therefore not representative of those commonly adopted in the industry or in the literature dedicated to HSP:

n number of jobs;

n_i number of treatments in the processing sequence of job i $(1 \le i \le n)$;

mc number of tanks in the processing line involved;

M_k tank k based on tank numbering set beforehand $(1 \le k \le mc)$.

NOTE.– In general, loading and unloading stations are assimilated to fictitious tanks respectively numbered *0* and $mc+1$. By convention, we consider that a carrier arrival in the system corresponds to the end of its loading, whereas its exit date corresponds to the time when the hoist places it at the unloading station. These commonly used (except in the case of extensions taking into account different possible configurations for loading and unloading stations) hypotheses will be the ones adopted in this chapter. Consequently, n_i does not include loading or unloading operations, which are implicitly assimilated to indexes $j = 0$ and $j = n_i + 1$.

Also:

$[i, j]$ the tank where the j^{th} treatment of a job i is performed, $(1 \le i \le n, 1 \le j \le n_i)$,

p_{ij}^s the actual time for the j^{th} treatment of a job i $(1 \le i \le n, 1 \le j \le n_i)$,

$\underline{p}_{ij}^s$ and $\overline{p}_{ij}^s$ maximum and minimum p_{ij}^s times as specified in the processing specifications,

$(i, j)^t$ transport of carrier i between $[i, j]$ and $[i, j+1]$,

p_{ij}^t time required to execute $(i, j)^t$, $(1 \le i \le n, 0 \le j \le n_i)$,

t_{ij}^t $(i, j)^t$ start date, therefore end of the treatment of i in $[i, j]$,

v_{kl} constant duration of empty hoist movement between M_k and M_l.

Modeling of the cyclic problem also requires the definition of:

– π the desired cyclic scheduling period;

– N the number of carriers to consider, i.e. the number of carriers simultaneously present in the system ($1 \le N \le N_{max}$, N_{max} being a constant representing the highest possible number). N can either be a data set beforehand or a variable defined at the same time as scheduling.

8.4.2. *CHSP resolution: basic problem*

The two most widely used models to solve the basic problem (see section 8.3.2.) are presented in this section, and are based on descriptions from Manier in 1994 [MAN 94a] and Lamothe in 1996 [LAM 96a]:

– the first model presented, called the "*carrier model*", involves all operations relative to carriers which can simultaneously be present in the system;

– the second, called the "*period model*", indicates constraints linking operations involved during a given period π cycle.

First, notations are simplified using the assumptions of the basic problem. The expression of constraints is given for each model and an example illustrates some associated solving methods, before presenting a common formalism [HAN 93].

8.4.2.1. *Simplification of notations*

The carriers, all identical, have the same number of treatments n_i, but this notation will not be simplified to avoid confusion with n, the number of jobs. In addition, all tanks are single-capacity and mono-function, and thus can be numbered in relation to the processing sequence order ([8.1]). Finally, all carriers have real identical processing times and transport time is independent of the moved carrier since departing and arriving tanks are the same ([8.2]).

$$\forall i, j, 1 \le i \le n, 1 \le j \le n_i, [i, j] = M_j \qquad [8.1]$$

$$\forall i, j, 2 \le i \le n, 1 \le j \le n_i, p_{ij}^s = p_{1j}^s \qquad [8.2]$$

$$\forall i, j, 2 \le i \le n, 0 \le j \le n_i, p_{ij}^t = p_{1j}^t$$

These assumptions, as used here and by the majority of authors, allow us to note:

$[j]$ the tank in which j^{th} processing for all jobs is done,

p_j^s the real duration of j^{th} processing for all jobs,

p_j^t transport $(i, j)^t$ time for all jobs.

On the other hand, the 1-periodic assumption requires that a carrier i enters the system exactly i-1 periods after carrier 1. In addition, all transports required by the processing specifications must be executed within a cycle ([8.3]), and variables t_{1j}^t can be renamed t_j for conciseness:

$$\forall i, j, 2 \le i \le n, 0 \le j \le n_i, \; t_{ij}^t = t_{1j}^t + (i-1)\pi \qquad [8.3]$$

8.4.2.2. *Carrier model*

$n_i + 1$ variables of this model are starting transport dates for carrier 1 and period π. They are calculated given that $t_0 = 0$ and considering three types of constraints:

– chemical constraints, i.e.:

- the no-wait succession of processing operations, indicating that the time between the date at which carrier 1 is lifted from tank $[j-1]$ and the date at which carrier 1 is placed in $[j]$ (corresponding to two consecutive operations) equals the sum of the transport time between $[j-1]$ and $[j]$ and the real processing time in $[j]$:

$$\forall j, 1 \le j \le n_i, \; t_j = t_{j-1} + p_{j-1}^t + p_j^s \qquad [8.4]$$

- respect for processing time limits:

$$\forall j, 1 \le j \le n_i, \underline{p}_j^s \le p_j^s \le \overline{p}_j^s$$

Note that these constraints are independent once more of i. Therefore, they only need to be expressed for carrier 1. In addition, the conjunction of the two previous equations leads to the global expression of chemical constraints by:

$$\forall j, 1 \le j \le n_i, \underline{p}_j^s \le t_j - t_{j-1} - p_{j-1}^t \le \overline{p}_j^s \qquad [8.5]$$

– unit capacity constraints for each mono-function tank which, combined with the 1-cyclic assumption, require that a carrier be lifted from each tank and that another take its place (but not necessarily in this order) during a cycle. π must therefore be higher than all actual processing times:

$\forall j, 1 \le j \le n_i, \pi > p_j^s$ or according to [8.4]: $\forall j, 0 \le j \le n_i, \pi > t_{j+1} - t_j - p_j^t$ [8.6]

– constraints linked to the single capacity hoist which must have enough time between two transports to move between the tank where it has just placed a carrier and the one where it will pick one up. This is expressed with the help of disjunctive constraints linking transport operations in pairs:

$$\forall\, i,i',j,j',\ 1 \le i,\ i' \le n,\ 0 \le j,j' \le n_i\ ,\ \begin{cases} t^t_{i'j'} \ge t^t_{ij} + p^t_{ij} + v_{[j+1][j']} \\ \text{or} \\ t^t_{ij} \ge t^t_{i'j'} + p^t_{i'j'} + v_{[j'+1][j]} \end{cases} \qquad [8.7]$$

By using equation [8.3]:

$$t^t_{i'j'} = t_{j'} + (i'-1)\pi \quad \text{and} \quad t^t_{ij} = t_j + (i-1)\pi$$

expression [8.7] can therefore be replaced by:

$$\begin{cases} t_{j'} + (i'-1)\,\pi \ge t_j + (i-1)\,\pi + p^t_j + v_{[j+1][j']} \\ \text{or} \\ t_j + (i-1)\,\pi \ge t_{j'} + (i'-1)\,\pi + p^t_{j'} + v_{[j'+1][j]} \end{cases}$$

In practice, disjunctions linking $(1, j')^t$ to $(i, j)^t$ are the same as those linking $(1+q, j')^t$ to $(i+q, j)^t$ (for q going from 1 to n-i, always by applying [8.3]). Disjunctions linking the first carrier to all the others simply have to be expressed. In addition, only N carriers simultaneously present in the system are liable to come into conflict for the use of the hoist. Consequently, only the disjunctions relative to the first N carriers must really be there. Finally, certain disjunctions are already arbitrated, as indicated in the next inequality:

$$\forall\, i, j, j', 1 < i \le N, 0 \le j' \le j \le n_i\ ,\ t_{j'} \le t_j + (i-1)\pi \qquad [8.8]$$

which expresses, for all j and j' operations such that j' precedes j in the processing specifications, that the first carrier will be lifted from $[j']$ before any i carrier entered later is itself lifted from $[j]$. This is easily justified in the mono-product and mono-function case: since all carriers have exactly the same processing specifications and visit each tank only once, their order of circulation in tanks is identical to the one in which they enter the system. In this way, the first carrier will get out of any $[j']$ tank before any other carrier i entered in the system later, and,

even more important, before i gets lifted from $[j]$ (since j' precedes j in the processing specifications). Finally, hoist constraints are entirely represented by:

$$\forall\, i, j, j',\ 2 \le i \le N,\ 0 \le j' < j \le n_i,\ \begin{cases} t_{j'} \ge t_j + (i-1)\,\pi + p^t_j + v_{[j+1][j']} \\ \text{or} \\ t_j + (i-1)\,\pi \ge t_{j'} + p^t_{j'} + v_{[j'+1][j]} \end{cases} \qquad [8.9]$$

These constraints can also be expressed by considering the period beginning with transport $(i, j)^t$ (of carrier i between $[j]$ and $[j+1]$), and ending when transport of carrier $i + 1$ between the same tanks starts. In this cycle, $(i, j)^t$ must be finished and the robot must have had time to move while empty before another carrier i' is transported between tanks $[j']$ and $[j'+1]$. The constraints are then independent from i and i' carrier numbers (Figure 8.6) and can be written as:

$$\forall\, j, j', 0 \le j < j' \le n_i,\ \alpha_{jj'} \le (t_{j'} - t_j) \bmod \pi \le \pi - \alpha_{j'j} \qquad [8.10]$$

with $\alpha_{jj'} = p^t_j + v_{[j+1][j']}$ and $\alpha_{j'j} = p^t_{j'} + v_{[j'+1][j]}$ (where $u \bmod w$ represents the rest of the Euclidian division of u by w).

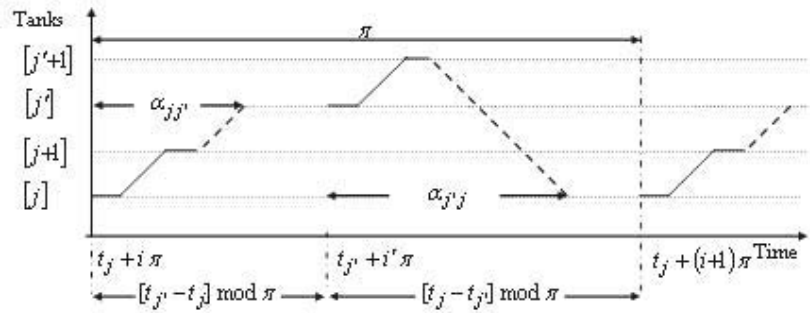


Figure 8.6. *"Hoist" constraints expressed during a cycle*

Function mod is non-linear. In addition, [HAN 93] introduces an integer representing the number of periods passed between beginning transport dates

respectively from $[j]$ to $[j+1]$ and $[j']$ to $[j'+1]$ ([8.11]) and expresses the following constraints linked to the hoist ([8.12]) and to integers ([8.13]).

$$y_{jj'}\,\pi \le t_{j'} - t_j \le (y_{jj'}+1)\,\pi$$

therefore, $(t_{j'} - t_j) \bmod \pi = t_{j'} - t_j - y_{jj'}\pi$, [8.11]

$$\forall\, j, j', 0 \le j < j' \le n_i, \alpha_{jj'} \le t_{j'} - t_j - y_{jj'}\pi \le \pi - \alpha_{j'j} \quad [8.12]$$

$$\forall\, j, j', 0 \le j < j' \le n_i,\ y_{jj'} + y_{j'j} = 1,\ y_{jj'} \in Z \quad [8.13]$$

The carrier model is thus defined by equations [8.2], [8.3], [8.5], [8.6] and [8.9] or [8.2], [8.3], [8.5], [8.6], [8.12] and [8.13]. It is widely used in other works, generally within a branch and bound search procedure which progressively builds scheduling. The branch and bound processes are different depending on the authors. Nevertheless, the solving principle common to all these approaches can be presented here in a generic way without having to detail each of them. Most often, it starts with a calculation limiting domains for t^{t}_{ij} and π by reducing the model to non-disjunctive constraints. Thus, a lower boundary π_{min} and a higher boundary π_{max} of π can be evaluated to initialize the search [MAN 94b]. π_{min} indicates that π must be at least higher than the minimal time required to accomplish all transports of a cycle on the one hand, and than the largest minimum processing time contained in the processing specifications on the other hand (constraints [8.6]). π_{max} is determined by the least productive scheduling (but one that definitely respects all constraints), which consists of processing all carriers consecutively, so that these carriers are never in the system simultaneously. The assessment of t_j domains is the direct consideration of constraints [8.3] and [8.5], from $t_0 = 0$. The following example illustrates this step. The line studied has three tanks, M_1, M_2 and M_3, and loading and unloading stations, M_0 and M_4. The maximum number of carriers present is set to be equal to 3, or one carrier per tank (no carrier should be in M_0 and M_4 because arrival in one of them corresponds either to arrival in the system or exit). According to the basic problem assumptions, the processing specifications follow tank indexing. Table 8.2 provides processing time boundaries in each one (in seconds).

	M_1	M_2	M_3
Minimum time	2	4	3
Maximum time	12	10	7

Table 8.2. *Processing time boundary in each tank*

Empty hoist movement duration between two adjacent tanks is equal to 1 second and the empty hoist movement duration between non-adjacent tanks is:

$$\forall\, j, j', 0 \le j \le 4, 1 \le j' \le 5, v[j][j'] = v[j'][j] = \left| j' - j \right|$$

Transport time is obtained by adding the time needed to lift and lower the carrier, i.e. 1 second. Any transport thus requires 2 seconds, because a carrier only moves between adjacent tanks. Period boundaries then become:

$$\pi \in]\pi_{min}, \pi_{max}\,], \pi_{min} = \max(4 \times 2,4) = 8, \pi_{max} = 2 + 2 + 2 + 4 + 2 + 3 + 2 + 4 = 21$$

and conjunctive constraints linked to processing times and to the 1-periodic cycle assumption provide the following domains:

$$t_0 = 0\ ;\ t_1 \ge t_0 + \underline{p_0^s} + p_0^t \text{ and } t_1 \le t_0 + \overline{p_0^s} + p_0^t \text{, where } t_1 \in [4,14]$$

similarly, $t_2 \ge 4 + 4 + 2$ and $t_2 \le 14 + 10 + 2$, where $t_2 \in [10, 26]$

finally, $t_3 \ge 10 + 3 + 2$ and $t_3 \le 26 + 7 + 2$, where $t_3 \in [15, 35]$.

π boundaries make it possible to deduce t^t_{ij} domains since they are linked to t_j by constraint [8.3] (Table 8.3).

Carrier	**(i, 0)^t**	**(i, 1)^t**	**(i, 2)^t**	**(i, 3)^t**
i = 2	[9, 21]	[13, 35]	[19, 47]	[24, 56]
i = 3	[18, 42]	[22, 56]	[28, 68]	[33, 77]

Table 8.3. *Variable domains for i > 1*

D_1: $(2,0)^t \prec (1,1)^t$ or $(1,1)^t \prec (2,0)^t$	D_{10}: $(3,2)^t \prec (1,3)^t$ or $(1,3)^t \prec (3,2)^t$
D_2: $(3,0)^t \prec (1,1)^t$ or $(1,1)^t \prec (3,0)^t$	D_{11}: $(4,2)^t \prec (1,3)^t$ or $(1,3)^t \prec (4,2)^t$
D_3: $(2,1)^t \prec (1,2)^t$ or $(1,2)^t \prec (2,1)^t$	D_{12}: $(2,1)^t \prec (1,3)^t$ or $(1,3)^t \prec (2,1)^t$
D_4: $(3,1)^t \prec (1,2)^t$ or $(1,2)^t \prec (3,1)^t$	D_{13}: $(3,1)^t \prec (1,3)^t$ or $(1,3)^t \prec (3,1)^t$
D_5: $(4,1)^t \prec (1,2)^t$ or $(1,2)^t \prec (4,1)^t$	D_{14}: $(4,1)^t \prec (1,3)^t$ or $(1,3)^t \prec (4,1)^t$
D_6: $(2,0)^t \prec (1,2)^t$ or $(1,2)^t \prec (2,0)^t$	D_{15}: $(2,0)^t \prec (1,3)^t$ or $(1,3)^t \prec (2,0)^t$
D_7: $(3,0)^t \prec (1,2)^t$ or $(1,2)^t \prec (3,0)^t$	D_{16}: $(3,0)^t \prec (1,3)^t$ or $(1,3)^t \prec (3,0)^t$
D_8: $(4,0)^t \prec (1,2)^t$ or $(1,2)^t \prec (4,0)^t$	D_{17}: $(4,0)^t \prec (1,3)^t$ or $(1,3)^t \prec (4,0)^t$
D_9: $(2,2)^t \prec (1,3)^t$ or $(1,3)^t \prec (2,2)^t$	

Table 8.4. *List of remaining disjunctions to arbiter*

At this step, some disjunctions are already arbitrated: those between transports relative to operations linked by a precedence relation in the processing sequence (constraint [8.8]) and those whose variable domains are separate (consequently imposing the order in which they will be executed). The tree search procedure then consists of progressively building scheduling by arbitrating remaining disjunctions (Table 8.4). The branching phase corresponds to arbitration of these disjunctions, and the bound phase adjusts previous domain limits and recalculates a lower bound of π for each partial solution (by still using constraints [8.3] and [8.5]). At each new arbitration, constraint system consistency must be checked by ensuring that the decision taken has not reduced any domains to the empty set and that all conjunctive constraints [8.3] and [8.5], as well as those corresponding to arbitrations already completed, are still satisfied.

The scientific studies contain several solving approaches (based on a principle similar to the one presented here), which are mainly distinguished by the arbitration strategy of disjunctive constraints (in other words the order in which they are considered) and evaluation and checking procedures used:

– in the Shapiro and Nuttle algorithm [SHA 88], the j^{th} tree level corresponds to the addition of carrier $j + 1$ (this is made up of all possible partial sequences between operations of carrier 1 and of the other carriers scheduled until $j + 1^{th}$) and the evaluation uses the resolution of several linear programs;

– the evaluation phase from Lei and Wang [LEI, 89a] adjusts variable domains (called "time windows"), and these authors use a specific matrix to gradually build scheduling by choosing the next transport to place (Figure 8.7);

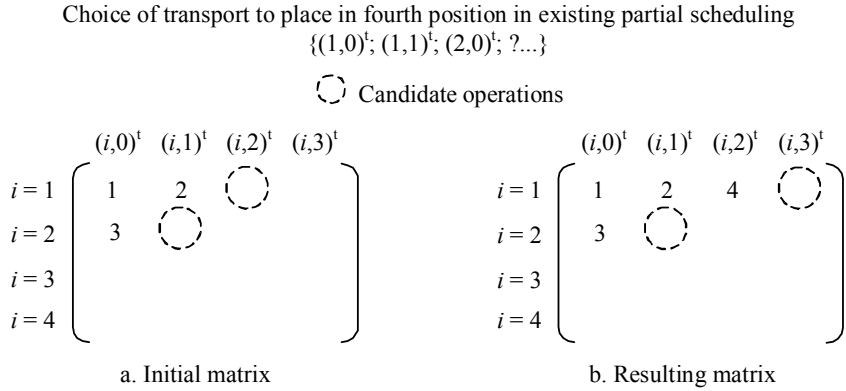
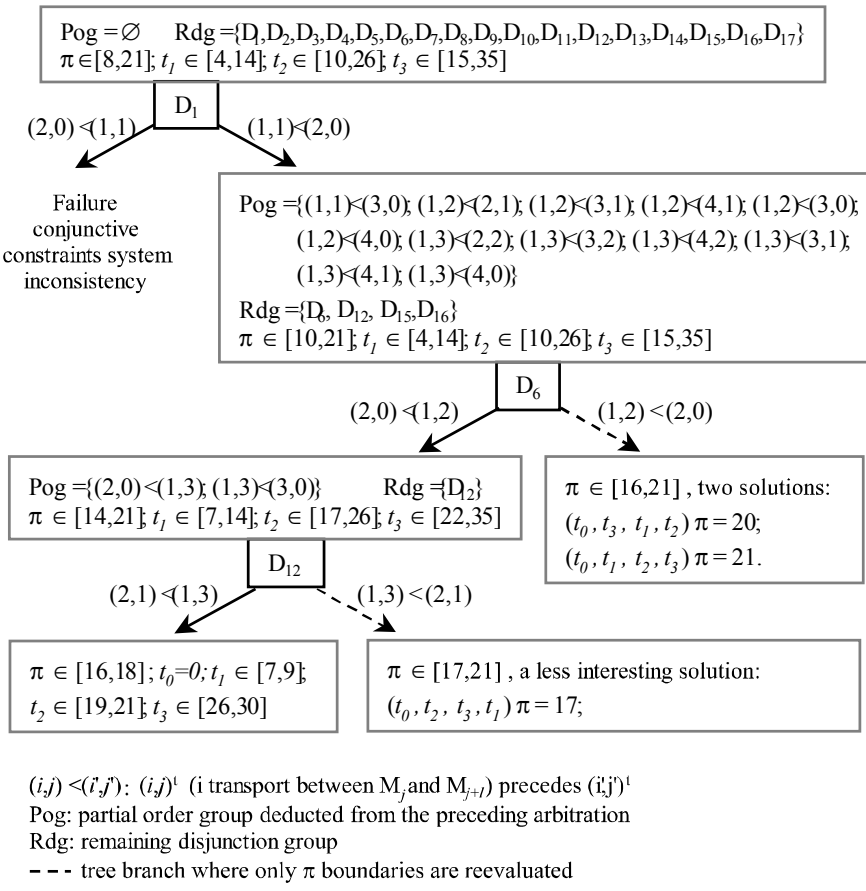


Figure 8.7. *Illustration of the exploration strategy from [LEI 89a]*

– [MAN 94b] uses constraint logic programming. Disjunctions are arbitrated starting with those for which there is only one possible choice, then by arbitrating those where one of the choices seems to have been more "probable" (by comparison with temporal windows involved), before ending by those where temporal windows provide no liable element to guide the choice. The evaluation uses a constraint propagation mechanism inherent to the languages used.



$(i,j) \prec (i',j')$: $(i,j)^t$ (i transport between M_j and M_{j+1}) precedes $(i',j')^t$
Pog: partial order group deducted from the preceding arbitration
Rdg: remaining disjunction group
– – – tree branch where only π boundaries are reevaluated

Figure 8.8. *Resolution using the carrier model*

To illustrate these methods in a concise way, the previous example is solved here with a simplified arbitration strategy which considers the disjunctions according to the order of Table 8.4 and gives priority to the carrier entered last in the system. The corresponding search tree is presented in Figure 8.8. A node corresponds to the arbitration of one of the disjunctions and also contains information available before branching, that is, the list of disjunctions remaining to arbitrate, as well as the list of partial orders and temporal windows deducted from the previous node arbitration. In fact, in order to limit the number of nodes, the evaluation phase is supposed to use a constraint propagation mechanism similar to the one from [MAN 94b]. π and t_j domains are the only ones indicated, knowing that the other domains deduct themselves as in the preliminary calculation.

The first branch thus built leads to a failure. In fact, since the line only contains one hoist, starting the second carrier before taking out the first one from M_1 would come down to placing carrier 2 before it is available. The procedure therefore translates the second D_1 alternative. The evaluation adjusts temporal windows by using the initial conjunctive constraints ([8.3] and [8.5]), to which the conjunctive constraint translating the order adopted for D_1 is added. The lower π boundary in particular increases by two and several partial orders are imposed by constraint propagation. Only four disjunctions remain to be arbitrated and the following node consists of processing the first one. The branch developed first again gives priority to carrier 2 and no inconsistency is detected. Boundaries are readjusted, two partial orders are taken out and only D_{12} remains to be arbitrated. By reiterating the process, a first solution is determined. It is the cycle represented by the generic transport sequence $\{(i, 0)^t; (i-1, 2)^t; (i, 1)^t; (i-1, 3)^t\}$, with a minimum period equal to 16 seconds. Retained execution dates are minimum t_j boundaries. The procedure then searches for a better solution by considering the other possible choice for D_{12}. The evaluation of the minimum π boundary turns out to be too high and the branch is forsaken. Figure 8.8 indicates that by continuing the evaluation, another less interesting solution would have been determined. Similarly, the evaluation of π_{min} for the second alternative of D_6 leads to forsaking the branch. The first solution found thus turns out to be the optimal solution. Note that because of the simplicity of the example involved, all the cases possibly encountered have not been illustrated. Notably, the only abandonments carried out here are linked to the impossibility of developing the choice retained for D_{12} or the fact that π_{min} is more important than the value of π for the first solution found. However, more generally, the exploration of a branch can also be forsaken if the choice obtained is possible, but the propagation detects an inconsistency (in other words, at least one disjunction no longer offers a choice) among remaining partial orders to be arbitrated.

[MAT 00] proposed a solving algorithm based on graph treatment for bound calculation. Some of the calculated bounds first enable us to check the feasibility of any solution before performing its complete evaluation. This enables us to save some calls to the evaluation procedure and therefore increases the performances of the branch and bound procedure.

8.4.2.3. *Period model*

The variables in this model are period π and transport starting dates throughout a cycle. We note as:

τ_j transport starting date between $[j]$ and $[j+1]$ in the cycle:

$$\forall\, j, 0 \le j \le n_i, \tau_j = t_j \bmod \pi\ .$$

Boolean variables $x_{jj'}$ are defined by:

$$\forall j, j',\ 0 \le j \le n_i,\ 0 \le j' \le n_i,\ x_{jj'} = 1 \text{ if } \tau_j > \tau_{j'},\ x_{jj'} = 0 \text{ otherwise, and } x_{jj'} + x_{j'j} = 1$$

$$\text{Therefore, } \forall j, j', 0 \le j \le n_i, 0 \le j' \le n_i, (t_{j'} - t_j) \bmod \pi = \tau_{j'} - \tau_j + x_{jj'}\pi,$$

and the constraints expression is consequently modified. Constraint [8.3] becomes useless because only one period is considered and equations [8.5], [8.6] and [8.9] or [8.10] become respectively:

$$\forall j, 1 \le j \le n_i, \underline{p}_j^s \le \tau_j - \tau_{j-1} - p_{j-1}^t + x_{j-1\,j}\pi \le \overline{p}_j^s \quad [8.14]$$

$$\forall j, 1 \le j \le n_i, \pi > \tau_{j+1} - \tau_j + x_{jj+1}\pi - p_j^t \quad [8.15]$$

$$\forall j, j', 0 \le j \le n_i, 0 \le j' \le n_i, \tau_{j'} - \tau_j + x_{jj'}\pi \ge p_j^t + v[j+1][j'] \quad [8.16]$$

Once more, different resolution approaches exist which are mainly distinguished by the definition of Boolean variables and the way in which they are determined:

– Phillips and Unger [PHI 76] use complete mixed programming to solve a manufacturing detailed example with 13 tanks which almost constitutes the only reference example for CHSP; their algorithm was still recently reused and extended to the duplicated tank case [ZHO 03];

– [LEI 93a] assume that the transport order is known in a cycle. The authors define π boundaries and use an iterative procedure determining if there is a feasible schedule for the different values belonging to this interval. This algorithm is reused by Lim to evaluate chromosomes in a genetic algorithm where solutions are represented by sequenced lists of tanks unloaded by the hoist in a cycle and where the strength depends on the obtained period [LIM 97];

– [LEI 94] and [ARM 94] define branch and bound procedures which gradually build scheduling. The [LEI 94] algorithm will be the only one detailed here (Figure 8.9), but it will enable the reader to understand what separates these approaches from those described in section 8.4.2.2. A node represents a partial transport order already placed in the cycle. The first two are associated with the arrival of a carrier in the system (marking the start of the cycle) and with the addition of M_1 to M_2 transport. Transports are then consecutively added. The width of a tree at a given level corresponds to the number of places possible in the sequence for the added operation by still keeping the arrival of the carrier in first place. At each node, the procedure evaluates π by default. The exploration strategy is in depth first and gives priority to the node with the minimal boundary. At level 2 of the tree presented in

Figure 8.9, operation τ_2 is added and the first branch explored has the lowest π boundary. Then the different possible places for τ_3 are considered. The first one leads to failure because the maximum processing time in M_3 is not respected. The two following places give solutions of respectively 17 and 16 second periods. A return to level 2 is carried out to search for a better solution. The evaluation of π by default does not prohibit exploration, but it is too high for two of the son nodes and the third son node leads to inconsistency, which proves the period-16 solution is optimal.

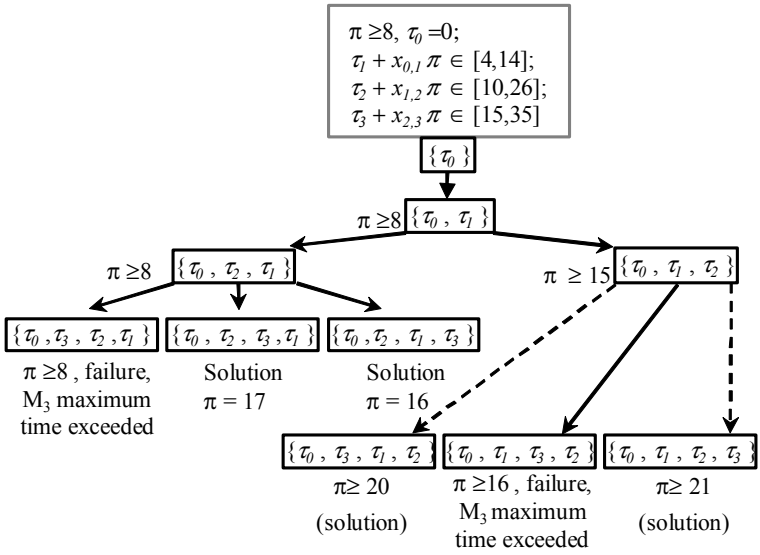


Figure 8.9. *Resolution using the period model [LEI 94]*

8.4.2.4. *Common formalism*

Hanen and Munier propose a common formalism by dividing all constraints by π [HAN 93]. This model uses given $L_{jj'}$ lengths and $H_{jj'}$ heights (which are integers):

$$\forall j, j',\ 0 \le j \le n_i,\ 0 \le j' \le n_i,\ V_{j'} - V_j \ge \frac{1}{\pi} L_{jj'} - H_{jj'}, \text{ with } V_j = \frac{\tau_j}{\pi} \quad [8.17]$$

These authors thus group all constraints in a bi-valued graph where peaks s_j represent V_j. The first disparity of [8.14] and constraints [8.16], for example, are equal respectively to the following expressions:

$$\forall j,\ 1 \le j \le n_i,\ L_{j\ 1j} = \underline{p}^s_j + p^t_{j\ 1} \text{ and } H_{j\ 1j} = 0$$

$$\forall j, j',\ 0 \le j \le n_i,\ 0 \le j' \le n_i,\ L_{jj'} = p^t_j + v_{[j+1][j']} \text{ and } H_{jj'} = x_{jj'}$$

This formalism also makes it possible to represent the carrier model by retaining length definitions and by defining V_j variables from t_j, and $H_{jj'}$ heights from $y_{jj'}$. The authors then demonstrate that the problem is a part of the central scheduling repetitive problem when $H_{jj'}$ are set. They present two tree search procedures by branch and bound. One, based on a carrier model, adjusts H and L boundaries at each node, thus evaluating π by default. The other uses a period model and gradually builds a solution (called "pattern") by listing transport permutations in a cycle. The default π evaluation is a critical path calculation in the diagram. Other authors [CHE 98] propose two branch and bound search procedures based on this model. One lists initial carrier distributions in the tanks, limiting their number by calculating a higher boundary of the number of carriers which can be lifted by the robot in a cycle. The algorithm verifies that there is an achievable scheduling for each initial state and evaluates π by default by solving a linear problem, obtained by relaxing certain constraints and represented in a bi-valued graph. The other procedure then determines the corresponding optimal transport sequence.

8.4.3. *Extensions*

Different extensions can be added to the basic models already described:

– search for cycles with a higher degree than 1 has in particular been studied by [MAN 94b, LEI 89a] for the carrier model and by [LEI 94] for the period model. The extension of chemical constraints and of constraints linked to the robot does not raise any specific problem; the extension of tank resource constraints is more significant. Globally, the same formalism can be retained and almost all tree search approaches can be generalized (except for the one from [ARM 94], which is based on the 1-periodic assumption to evaluate π by default). However, the important number of variables generally limits these approaches to a maximum of three or four degree solutions;

– the consideration of multi-capacity tanks (called duplicated tanks) has been addressed in particular by [SHA 88, LEI 89a, HAN 93, MAN 94b, LEI 94, LIU 02, RIE 02, ZHO 03]. Generally, it does not lead to additional constraints, but a modification of constraints [8.6] or [8.15], consisting of the meaning that a carrier can only enter a tank containing b_k places on the condition that one of the b_k carriers preceding it has been removed. The authors generally do not give prior place assignment and consider the longest transport distance (see section 8.2.1);

– the study of all possible configurations for loading and unloading stations, by taking into account the duration of these operations and of cleaning operations in the processing specifications was conducted by [SHA 88] and [MAN 94b], adding additional constraints for this purpose. The first one defined the notion of carrier circulation, specific constraint involving that the carrier unloaded during a cycle be immediately reloaded and reused as arriving carrier in the next cycle;

– the generalization of multi-hoist lines, developed for the approaches by [LEI 89a, HAN 93, ARM 94, MAN 94b], respectively in [LEI 91, HAN 94, LEI 93b, BAP 96, MAN 00, YAN 01, RIE 02]. All these approaches share the line in zones each served by one of the hoists. Contrary to those from [LEI 91] (limited to 2 hoists) and from [LEI 93b], the extension of [MAN 94b] ([MAN 00]) considers zones which may be disjoint and/or continuous. A series of transports, and not a work zone, is assigned to each hoist, which encourages more flexible management while balancing the hoist workload. However, this increases the risk of collision between hoists, and new constraints (called "spatial constraints") are used to avoid interferences. Expressed in the form of disjunctive rules, they were deducted from the analysis of different interference configurations liable to appear. The notion of hoist clearing (section 8.2.2) and congestion are also considered. This approach was also completed by improving the study of transport assignment for hoists [VAR 97]. In [HAN 94], temporal and spatial constraints linked to the multi-hoist case are added to the hybrid model in section 8.4.2.4 by using binary variables h_{ij} to represent arbitration of corresponding disjunctions in the diagram, and then by resolving a linear problem for each occurrence of these variables.

More recently, and for a given number of hoists, the authors of [YAN 01] use a simulated annealing algorithm to find the optimal partition in zones. Nevertheless they still make the same assumption as [LEI 91] (continuous zones with a single boundary station between two contiguous zones).

The authors of [RIE 02] hybridize constraint logic programming and mixed integer programming to solve the multi-hoist case either assigning disjoint zones like [LEI 91] or using collision-based assignment like [MAN 00]. A preprocessing step is used to eliminate certain contradictory situations in advance, which makes it possible to greatly reduce the number of Boolean variables used to represent the order in which any hoist performs transports;

– the study of both design and scheduling of electroplating facilities. [ARM 96] search for the best number of hoists, but also with the same partition approach as [LEI 91]. [MAN 06, MAN 08] propose a decision support system based on an evolutionary approach, a linear programming evaluation model, and simulation. They *a priori* do not fix either the number of hoists, or the assignment of transport operations to each hoist. They determine schedules with the best couples (cycle time, number of hoists), in order to help a designer to choose the best compromise among the solutions generated.

8.4.4. *Multi-product case*

The authors who have focused on the multi-product CHSP [PTU 95, VAR 00, MAT 06] have broken it down into several mono-product cyclic problems. They then consider as many sub-problems as different processing specifications to be treated. Ptuskin supposes that the mixing (order of the different types of jobs in the arrival sequence) is known. He determines their exact arrival date within this sequence and real processing times to minimize the period. [VAR 00] does not make this assumption and searches for a generic minimal period cycle to make a carrier enter the line every π units of time, regardless of its processing specifications among those authorized in the mixing. In both cases, the solution needed has a common period at every mono-product sub-problem. Ptuskin uses an iterative procedure and Varnier proposes a branch and bound procedure to arbitrate the disjunctions between different carriers, according to a principle relatively similar to that in section 8.4.2.2. Finally, [MAT 06] extends the solving approach proposed in [MAT 00] to tackle the multi-product CHSP. These authors consider two types of products. They use fictitious tanks to transform the 2-product CHSP which processing sequence contains mt treatments in a mono-product CHSP which processing sequence contains $2mt$ treatments. Type 1 products only visit tanks with odd numbers whereas type 2 products only visit the tanks with even numbers. This implies the addition of new arcs in the graph representing the constraints relative to type 2 products. The solving algorithm can then treat the graph in the same way as in the mono-product case.

8.5. Resolution of other problems presented

The resolution of non-cyclic problems is generally based on a linear carrier model in which simplifications presented in section 8.4.2.1 are no longer relevant, because the notion of period no longer exists and several processing specifications are considered. The constraints used are based on the same formalism, but must be expressed for all carriers, since the first one no longer plays a specific role. In this way, chemical constraints ([8.5] type) must be expressed for all carriers, constraints [8.6] involving the period are replaced by disjunctive constraints somewhat similar to constraints [8.7] expressed for the hoist and are applied with no modification.

8.5.1. *Optimization of temporary phases*

[VAR 96] optimizes the transient phase resulting from the transition of an initial cycle A to a final cycle B. This author takes advantage of the degraded production mode for each cycle (because of a load decrease for type A jobs and an increase for B) to temporarily combine transports from both cycles into a single forecasted

scheduling. A and B cycles are elements of the problem. Once more, the proposed resolution method is similar to the one illustrated in section 8.4.2.2 and also uses constraint logic programming. However, the solution needed is not cyclic, and the criteria to minimize are transient phase durations. A branch and bound search procedure makes it possible to arbitrate disjunctive constraints linked to resource sharing. A lower and upper criteria boundary is determined beforehand and to limit the number of disjunctions, partial orders of A and B cycles are kept.

8.5.2. *Job scheduling at line arrival*

Faced with a much higher number of variables, the authors addressing this problem use greedy progressive construction algorithms or local search methods to generate job arrival sequences, and then determine for each one if achievable transport scheduling exists with the help of simulation or dedicated heuristics. In the case of a greedy method [CAU 95, FLE 95], the new job to add in the developing sequence must be determined at each iteration. It is the non-scheduled job whose earliest beginning date is the lowest. This date is evaluated by an iterative procedure using a discrete event simulator, which plays the same role as the conjunctive constraint system verification in section 8.4.2.2. It is equivalent to calculating time windows of transports by expressing chemical constraints [8.5] for all the carriers involved in the partial solution. It does not arbitrate disjunctions over resources (type [8.6] and [8.7]), but detects possible overlapping between time windows. In this case, the iterative procedure delays the job arrival to solve these conflicts. When a local search algorithm is used, it provides a carrier a beginning sequence for simulation so that it can determine beginning dates by an iterative procedure similar to the previous one. [CAU 95 and FLE 95] combine a stochastic descent with the simulator already presented, whereas [LAC 98] uses a kangaroo algorithm, associated with a simulation based on a multi-agent model, which can also be used for real-time hoist control and considers carriers as a resource.

In the same way but for an extended problem, the authors of [FLE 01] use several tools to solve a problem in which random events imply variations in transportation times. They call it the stochastic HSP (SHSP). They aim at minimizing the consequences of such variations in terms of increase of the makespan and number of job damages due to the violations of time windows constraints. They propose a model which again associates stochastic metaheuristics to manage the production at the system entrance, decision rules to manage the inner production, and simulation to evaluate the criterion to minimize. [CAU 97] and [BLO 99a] have a methodology that is almost similar by combining respectively simulated annealing and tabu search into a dedicated heuristic. In fact, since the problem is over-constrained, the simulation manipulates numerous solutions in order to find a few (or none) achievable. Replacing it by a heuristic which builds an

achievable scheduling from the beginning sequence proposed may turn out to be a more efficient solution. The heuristics proposed on this subject by [CAU 97] gradually build the transport sequence by considering jobs consecutively according to the imposed arrival sequence and by delaying the arrival date of the last job in the partial sequence if performing one of the associated transports creates a time window overlap. [BLO 99a] uses a procedure inspired by the bottleneck machine heuristic [ADA 88] (also see Chapter 2) modified to consider constraints specific to HSP (notably bounded processing times and no-wait constraints) [ROS 99]. The goal is to use the scheduling of job entries as a control to limit resource conflicts, deadlocks and maximum processing time constraint violations (see section 8.3.3). Imposing job entry dates that were judiciously calculated makes it possible to significantly reduce the number and magnitude of observed violations of constraints.

8.5.3. *DHSP resolution*

DHSP resolution methods are heuristic, and are mostly strongly based on solving principles encountered to date:

– Yih uses an iterative procedure which attempts to enter the new carrier as soon as possible [YIH 94], calculates its transports' execution time windows by using chemical constraints translating the minimum operation times, and then attempts to use certain available processing time margins or delays the entrance of this job if necessary, when overlaps between these time windows and those of the previous carriers are detected;

– Ge and Yih, Cheng and Smith, and Lamothe all present tree search procedures to progressively construct a schedule [GE 95, CHE 95, LAM 96a]. The first one is similar to the one described in section 8.4.2.3, because each node represents a partial transport sequence, and each level corresponds to the addition of a transport by prioritizing the new carrier whenever possible and by choosing the carrier transport which is the closest to its maximum processing time otherwise. Solving a linear program combining conjunctive constraints of the "carrier" model and those indicating the partial sequence makes it possible to check the consistency of the constraints system for each sub-branch. The tree is deeply analyzed and the algorithm stops at the first determined achievable scheduling. The other methods are closer to the ones illustrated in section 8.4.2.2, because a node corresponds to the arbitration of a disjunction. These methods use a graph to represent conjunctive constraints associated with a partial solution (chemical constraints and constraints from previous arbitrations). In this graph, calculating the longest route enables us to adjust transport execution time windows and to detect possible inconsistencies by positive circuit detection. They differ, however, by their exploration strategy and their stop criteria. The first one [CHE 95] uses dominance rules from the traditional

constraint analysis theory [ERS 80] to identify trivial arbitration and quickly decrease research space. It uses a heuristic rule when indecision remains (see section 5.4.2) and stops without backtracking in the case of inconsistency (thus offering no solution). The second one [LAM 96a] arbitrates disjunctions linked to tanks with priority and by increasing operation start dates. It is complete and backtracking is more powerful than traditional backtracking since a portion of information from the previous resolution is kept. In addition, this author is the only person to our knowledge who has solved the DHSP for complex lines, similar to industrial facilities;

– [ROS 99] uses an evaluation procedure of the longest routes in a graph similar to [CHE 95] and [LAM 96a] to evaluate transport sequences. However, these are not partial sequences evaluated at each node of a tree search procedure; these are complete sequences, represented in the form of chromosomes in a genetic algorithm covering the search space. The strength of these solutions is proportional to the end date for all jobs, and they are penalized when an inconsistency is detected;

– Spacek *et al*. model the problem in an inventive way in the form of a P-temporal Petri network [SPA 99]. The variables are transition launching dates. This representation is illustrated in the form of a state equation in algebra (max, +). The components of the state represent execution dates for processing vector $x(k)$ and carrier transport operations k. Determining a solution $x^{\#}(k)$ is made possible by the resolution of a problem obtained by relaxing the hoist's unit capacity constraints. Then, the algorithm searches for overlaps of time windows linked to the hoist and resolves them by modifying the control vector $u(k)$ according to a "repair" procedure inspired from [YIH 94];

– similarly to [CHE 95], Hindi and Fleszar use a non-standard CSP (Constraint Satisfaction Problem) model [HIN 04]. The variables used in this model correspond to hoist operations. At each step, one of the available hoist operations is chosen and scheduled at the earliest possible time, while avoiding all conflicts. The choice made gives priority to the operation with the earliest start time (if possible). In case of infeasibility (if an upper bound is not respected for soaking durations), then the considered transport operation is delayed and backtrack is initiated (another operation is chosen according to this strategy);

– in [PAU 07], the jobs are picked up one after the other according to the entry sequence. Then the activities of a job are consecutively scheduled. Once a job is introduced, all its activities are planned for execution within corresponding time windows. These time windows are continuously adapted, then the activities associated with one job can be re-scheduled but not re-sequenced at a later step.

8.5.4. *RHSP resolution*

Real-time control has motivated several studies on the real-time assignment of transports to hoists. There are two main approaches: one is based on heuristic rules included in an expert system or within list algorithms, the other using state change probabilities.

Thesen and Lei develop an expert system which chooses the heuristic to use according to the current line state [THE 90]. Decision rules enable hoists to adapt in real time to the situation. Four allocation rules are used. The first one defines work zones beforehand to balance the load between robots. The second one accomplishes this balance in real time by dynamically readjusting work zone limits. The third one allocates each transport to the robot whose current position is the closest to the starting tank and the last prioritizes the hoist which is at the farthest left of loading tasks and applies the third for the other transports.

[YIH 93] uses both neural networks and semi-Markov decision models. The latter describes some of the possible system states. A transition matrix is built by studying the loss or gain associated with a decision. This approach is then completed by using a network or neurons to generalize the policy obtained by resolving the semi-Markov model for all possible states.

Jégou *et al.* propose two multi-agent systems to solve the RHSP, considering each tank and each hoist as an agent [JEG 06]. The first system is used to determine the input date of the next job; the second system allows us to assign transfer operations to hoists and to schedule the actual operation of these hoists. For this goal, it uses the flexibility of the time windows.

8.6. Conclusion

The rapid overview of studies related to HSP provided in this chapter is not exhaustive. It could furthermore be linked with other problems, only varying by a few (or even only one) assumptions or constraints. The scheduling problem for arriving jobs then resembles the car sequencing problem in the automobile industry. Similarly, establishing certain parallels with the Crane scheduling problem [LIE 82], vehicle routing with time windows [DUH 97] or simultaneous scheduling of machines and material handling resources in flexible production systems [ULU 97] could also have been informative. Finally, other problems were considered in the hoist system context. Certain authors focus on the influence of tank layout over shop productivity [GRU 97], and others discuss robustness of solutions in order to take the different production hazards into consideration [MAR 01, FAR 01]. Finally, many “variants” among the numerous variants we have identified still remain

interesting and have opened issues for researchers. Nevertheless, recent works also point out new research directions, by extending classical HSPs to consider stochastic events and their influence on the line [FLE 01], environmental issues [XU 04, KUN 06, SUB 06], or to solve both design and scheduling problems [MAN 06, MAN 08].

8.7. Bibliography

[ADA 88] ADAMS J., BALAS E. and ZAWACK D., "The shifting bottleneck procedure for job shop scheduling", *Management Science*, vol. 34, no. 3, p. 391-401, 1988.

[ARM 94] ARMSTRONG R., LEI L. and SHANHONG G., "A bounding scheme for deriving the minimal cycle time of a single-transporter n-stage process with time-window constraints", *European Journal of Operational Research*, vol. 75, p. 1-11, 1994.

[ARM 96] ARMSTRONG R., GU S. and LEI L., "A greedy algorithm to determine the number of transporters in a cyclic electroplating process", *Institute of Industrial Engineers Transactions*, vol. 28, no. 5, p. 347-355, 1996.

[BAP 96] BAPTISTE P., LEGEARD B., MANIER M.-A. and VARNIER C., "Résolution d'un problème d'ordonnancement avec la PLC", *European Journal of Automated Systems*, vol. 30, no. 2-3, p. 201-230, 1996.

[BLO 97] BLOCH C., BACHELU A., VARNIER C. and BAPTISTE P., "Hoist scheduling problem: state-of-the-art", *Fourth International Federation of Automatic Control Workshop on Intelligent Manufacturing Systems*, Seoul, Korea, p. 177-185, 1997.

[BLO 99a] BLOCH C., VARNIER C. and BAPTISTE P., "A taboo search combined with a modified shifting bottleneck heuristic for solving a blocking scheduling problem with bounded processing times", *15th Conference of the International Federation of Operational Research Societies*, Beijing, China, p. 125-126, 1999.

[BLO 99b] BLOCH C. and MANIER M.-A., "Notation and typology for the hoist scheduling problem", *Institute of Electrical and Electronics Engineers Systems, Man and Cybernetics Conference*, Tokyo, Japan, vol. 6, p. 475-480, 1999.

[CAU 95] CAUX C., FLEURY G., GOURGAND M. and KELLERT P., "Couplage méthodes d'ordonnancement – simulation pour l'ordonnancement de systèmes industriels de traitement de surface", *RAIRO-Recherche Opérationnelle*, vol. 29, no. 4, p. 391, 1995.

[CAU 97] CAUX C. and PIERREVAL H., "Solving a hoist scheduling problem as a sequencing problem", *International Federation of Automatic Control/International Federation for Information Processing Conference on Management and Control of Production and Logistics*, Campinas, Brazil, p. 372-376, 1997.

[CHE 95] CHENG C.-C. and SMITH S. F., "A constraint-posting framework for scheduling under complex constraints", *Symposium on Emerging Technologies and Factory Automation*, Paris (France), vol. 1, p. 269-280, 1995.

[CHE 98] CHEN H., CHU C. and PROTH J.-M., "Cyclic scheduling of a hoist with time window constraints", *Institute of Electrical and Electronics Engineers Transactions on Robotics and Automation*, vol. 14, no. 1, p. 144-152, 1998.

[DUH 97] DUHAMEL C., POTVIN J.Y. and ROUSSEAU J.M., "A tabu search heuristic for the vehicle routing problem with backhauls and time windows", *Transportation Science*, vol. 31, no. 1, p. 49-59, 1997.

[ERS 80] ERSCHLER J., ROUBELLAT F. and VERNHES J.-P., "Characterizing the set of feasible sequences for *n* jobs to be carried out on a single machine", *European Journal of Operational Research*, vol. 4, no. 3, p. 189-194, 1980.

[FAL 91] FALKENAUER E. and BOUFFOUIX S., "A genetic algorithm for job shop", *Institute of Electrical and Electronics Engineers International Conference on Robotics and Automation*, p. 824-829, 1991.

[FAR 01] FARGIER H., and LAMOTHE J., "Handling soft constraints in hoist scheduling problems: the fuzzy approach", *Engineering Applications of Artificial Intelligence*, vol. 14, p. 387-399, 2001.

[FLE 95] FLEURY G., "Applications de méthodes stochastiques inspirées du recuit simulé à des problèmes d'ordonnancement", *Automatique productique informatique industrielle*, vol. 29, no. 4-5, p. 445-470, 1995.

[FLE 01] FLEURY G., GOURGAND M. and LACOMME P., "Metaheuristics for the Stochastic Hoist Scheduling Problem (SHSP)", *International Journal of Production Research*, vol. 39, no. 15, p. 3419-3457, 2001.

[GE 95] GE Y. and YIH Y., "Crane scheduling with time windows in circuit board production lines", *International Journal of Production Research*, vol. 33, no. 5, p. 1187-1189, 1995.

[GRA 79] GRAHAM. R.L., LAWLER E.L., LENSTRA J.K. and RINNOOY KAN A.H.G., "Optimization and approximation in deterministic sequencing and scheduling theory: a survey", *Ann. Discrete Math*., vol. 5, p. 287-326, 1979.

[GRU 97] GRUNDER O., BAPTISTE P. and CHAPPE D., "The relationship between the physical layout of the work stations and the productivity of a saturated single-hoist production line", *International Journal of Production Research*, vol. 35, no. 8, p. 2189-2211, 1997.

[HAN 93] HANEN C. and MUNIER A., Ordonnancement cyclique d'un robot sur une ligne de galvanoplastie: modèles et algorithmes, LITP/IBP, Pierre and Marie Curie University, Paris, LITP research report 93.30, 1993.

[HAN 94] HANEN C., "Periodic scheduling of several hoists", *Fourth International Workshop on Project Management and Scheduling*, Leuven, Belgium, p. 108-110, 1994.

[HIN 04] HINDI K. S. and FLESZAR K., "A constraint propagation heuristic for the single hoist, multiple products scheduling problem", *Computers and Industrial Engineering*, vol. 47, p. 91-101, 2004.

[JEG 06] JÉGOU D., KIM D.-W., BAPTISTE P. and LEE KWANG H., "A contract net based intelligent agent system for solving the reactive hoist scheduling problems", *Expert Systems with Applications*, vol. 30, p. 156-167, 2006.

[KUN 06] KUNTAY I., XU Q., UYGUN K. and HUANG Y. L., "Environmentally Conscious Hoist Scheduling for Electroplating Facilities", *Chemical Engineering Communications*, vol. 193, p. 273-292, 2006.

[LAC 98] LACOMME P., Optimisation des systèmes de production: méthodes stochastiques et approche multi-agents, PhD thesis, Clermont-Ferrand, 1998.

[LAM 96a] LAMOTHE J., Une approche pour l'ordonnancement dynamique d'un atelier de traitement de surface, PhD Thesis, Ecole Nationale Supérieure de l'Aéronautique et de l'Espace, Toulouse, 1996.

[LAM 96b] LAMOTHE J., THIERRY C. and DELMAS J., "A multihoist model for the real time hoist scheduling problem", *Computational Engineering in Systems Application Multiconference-Symposium on Discrete Events and Manufacturing Systems*, Lille, France, p. 461-466, 1996.

[LEI 89a] LEI L. and WANG T.-J., On the optimal cyclic schedules of single hoist electroplating processes, Rutgers University, Research report no. 89-0006, 1989.

[LEI 89b] LEI L. and WANG T.-J., A proof: the cyclic hoist scheduling problem is NP-complete, Rutgers University, Research report no. 89-0016, 1989.

[LEI 91] LEI L. and WANG T.-J., "The minimum common cycle algorithm for cycle scheduling of two material handling hoists with time window constraints", *Management Science*, vol. 37, no. 12, p. 1629-1639, 1991.

[LEI 93a] LEI L., "Determining the optimal starting time in a cyclic schedule with a given route", *Computers and Operations Research*, vol. 20, no. 8, p. 807-816, 1993.

[LEI 93b] LEI L., ARMSTRONG R and GU S., "Minimizing the fleet size with dependent time-window and single-track constraint", *Operations Research Letters*, vol. 14, p. 91-98, 1993.

[LEI 94] LEI L. and WANG T.-J., "Determining the optimal cyclic hoist schedules in a single-hoist electroplating line", *Institute of Industrial Engineers Transactions*, vol. 2, p. 25-33, 1994.

[LIE 82] LIEBERMAN R. W. and TURKSEN I. B., "Two-operation crane scheduling problems", *Institute of Industrial Engineers Transactions*, vol. 14, no. 3, p. 147-155, 1982.

[LIM 97] LIM J.-M., "A genetic algorithm for a single hoist scheduling in the printed-circuit-board electroplating line", *Computers and Industrial Engineering*, vol. 33, no. 3-4, p. 789-792, 1997.

[LIU 02] LIU J., JIANG Y., and ZHOU Z., "Cyclic scheduling of a single hoist in extended electroplating lines: a comprehensive integer programming solution". *IIE Transactions*, vol. 34, no. 10, p. 905-914, 2002.

[MAN 94a] MANIER M.-A. and BAPTISTE P., "Etat de l'art: ordonnancement de robots de manutention en galvanoplastie", *Automatique productique informatique industrielle*, vol. 28, no. 1, p. 7-35, 1994.

[MAN 94b] MANIER-LACOSTE M.-A., Contribution à l'ordonnancement cyclique du système de manutention d'une ligne de galvanoplastie, PhD Thesis, Besançon, 1994.

[MAN 00] MANIER M.-A., VARNIER C. and BAPTISTE P., "Constraint-based Model for the cyclic Multi-Hoists Scheduling Problem", *Production Planning and Control*, vol. 11, no. 3, p. 244-257, 2000.

[MAN 03] MANIER M.-A. and BLOCH C., "A Classification for Hoist Scheduling Problems", *The International Journal of Flexible Manufacturing Systems*, vol. 15, no. 1, p. 37-55, 2003, Kluwer Academic Publishers.

[MAN 06] MANIER M.-A. and LAMROUS S., "Design and scheduling of electroplating facilities", *IEEE International Conference on Service Systems and Service Management*, Troyes, France, p. 1114-1119, 2006.

[MAN 08] MANIER M.-A. and LAMROUS S., "An evolutionary approach for the design and scheduling of electroplating facilities", to appear in *Journal of Mathematical Modelling and Algorithms*, 2008.

[MAR 01] MARTEAU S., BONHOMME P., AYGALINC P. and CALVEZ S., "A control model for multi-products processing", *8th IEEE International Conference on Emerging Technologies and Factory Automation*, Antibes-Juan les Pins, France, vol. 1, p. 527-536, 2001.

[MAT 00] MATEO M., COMPANYS R. and BAUTISTA J., "Bounded cycle time for the cyclic hoist scheduling problem", *First World Conference on Production and Operations Management*, Seville, Spain, p. 1-10, 2000.

[MAT 06] MATEO M. and COMPANYS R., "Hoist scheduling in a chemical line to produce batches with identical sizes of different products", *Sixième Conférence Francophone de Modélisation et Simulation : Modélisation, Optimisation et Simulation des Systèmes (MOSIM'06)*, Rabat, Morocco, p. 677-684, 2006.

[PAU 07] PAUL HENRIK J., BIERWIRTH C., and KOPFER H., "A heuristic procedure for multi-item hoist production line", *International Journal of Production Economics*", vol. 105, p. 54-69, 2007.

[PHI 76] PHILLIPS L.W. and UNGER P.S., "Mathematical programming solution of a hoist scheduling program", *American Institute of Industrial Engineers Transactions*, vol. 8, no. 2, p. 219-225, 1976.

[PTU 95] PTUSKIN A.S., "No-wait periodic scheduling of non-identical parts in flexible manufacturing lines with fuzzy processing times", *International Workshop on Intelligent Scheduling of Robots and Flexible Manufacturing Systems*, Center for Technological Education, Holon Press, Holon, Israel, p. 210-222, 1995.

[RIE 02] RIERA D. and YORKE-SMITH N., "An Improved Hybrid Model for the Generic Hoist Scheduling Problem", *Annals of Operations Research*, vol. 115, p. 173-191, 2002.

[ROS 99] ROSSE-BLOCH C., Contribution à l'ordonnancement dynamique de lignes de traitement de surface, PhD Thesis, Besançon, 1999.

[SHA 88] SHAPIRO G.W. and NUTTLE H.L.W., "Hoist scheduling for a PCB electroplating facility", *Institute of Industrial Engineers Transactions*, vol. 20, no. 2, p. 157-167, 1988.

[SON 95] SONG W., STORCH R. L. and ZABINSKY Z. B., "An example for scheduling a chemical processing tank line", *Symposium on Emerging Technologies and Factory Automation*, Paris, France, vol. 1, p. 475-482, 1995.

[SPA 99] SPACEK P., MANIER M.-A. and EL MOUDNI A., "Control of an electroplating line in the Max and Min algebras", *International Journal of Systems Science*, vol. 30, no. 7, p. 759-778, 1999.

[SUB 06] SUBAI C., BAPTISTE P. and NIEL E., "Scheduling issues for environmentally responsible manufacturing: the case of hoist scheduling in an electroplating line", *International Journal of Production Economics*, vol. 99, p. 74-87, 2006.

[THE 90] THESEN A. and LEI L., "An expert scheduling system for material handling hoists", *Journal of Manufacturing Systems*, vol. 9, no. 3, p. 247-252, 1990.

[ULU 97] ULUSOY G., SIVRIKAYA-SERIFOGLU F. and BILGE U., "A genetic algorithm approach to the simultaneous scheduling of machines and automated guided vehicles", *Computers and Operations Research*, vol. 24, no. 4, p. 335-351, 1997.

[VAR 96] VARNIER C., Extensions du "Hoist Scheduling Problem" cyclique. Résolution basée sur un traitement des contraintes disjonctives en programmation en logique avec contraintes, PhD Thesis, Besançon, 1996.

[VAR 97] VARNIER C., BACHELU A., BAPTISTE P., "Resolution of the cyclic multi-hoist scheduling problem", *Information Systems and Operational Research*, vol. 35, no. 4, p. 309-324, 1997.

[VAR 00] VARNIER C., JEUNEHOMME N., "A cyclic approach for the multi-product HSP", *7th Workshop on Project Management and Scheduling*, Osnabrück, Germany, 2000.

[XU 04] XU Q. and HUANG Y., "Graph-Assisted Cyclic Hoist Scheduling for Environmentally Benign Electroplating", *Industrial and Engineering Chemistry Research*, vol. 43, no. 26, p. 8307-8316, 2004.

[YAN 01] YANG G.W., JU D.P., ZHENG W.M. and LAM K., "Solving multiple hoist scheduling problems by use of simulated annealing", *Ruan Jian Xue Bao/Journal of Software*, vol. 12, no. 1, p. 11-17, 2001.

[YIH 93] YIH Y., LIANG T.P. and MOSKOWITZ H., "Robot scheduling in a circuit board production line: an OR/ANN approach", *Institute of Industrial Engineers Transactions*, vol. 25, no. 2, p. 26-33, 1993.

[YIH 94] YIH Y., "An algorithm for hoist scheduling problems", *International Journal of Production Research*, vol. 32, no. 3, p. 501-516, 1994.

[ZHO 03] ZHOU Z. and LI L., "Single hoist cyclic scheduling with multiple tanks: a material handling solution", *Computers & Operations Research*, vol. 30, no. 6, p. 811-819, 2003.

8.8. Appendix: notation

	Reference	Notation
1	[ARM 94]	CHSP |19//diss|/21|Tmin
2	[ARM 96]	CHSP |mh,50,ct//diss|/52|mhmin
3	[BAP 96]	CHSP |3,18,10//|/19,recrc|Tmin
4	[BLO 99a]	PHSP|mh,mt,1/ret/|nparts/nps,nop|Cmax
5	[CAU 95]	PHSP |3,4,synchro,(15),(15),(15)//diss|16/24,17,recrc|Cmax
6	[CAU 97]	PHSP |16//diss|36/10,12,recrc|Cmax
7	[CHE 95]	DHSP |5//diss|100/100,7|Cmax
8	[CHE 98]	CHSP|13//diss|/14|Tmin
9	[FAR 01]	DHSP|mt/nc/|nparts/nop|(Tmin,Qmax)
10	[FLE 95]	PHSP |1,12,ct/nc/|20/nps,4,recrc|Cmax
11	[FLE 01]*	SHSP*| mh,16 / 6, empty/ diss | nparts / 12, nop| (Cmax, Other)
12	[GE 95]	DHSP |mt//diss|100/nps,nop|Cmax
13	[HAN 93]	CHSP |1,14,ct//diss|/16,recrc|Tmin
14	[HAN 94]	CHSP |mh,mt,ct//diss|/nop|Tmin
15	[HIN 04]	DHSP | 5 / / diss | 100 / 100,6| Cmax
16	[JEG 06]	RHSP|2,16,1//load-unload| nparts/14|(DefectiveJobsMin,ThroughputMax)
17	[LAC 98]	PHSP|16/6,empty/diss|nparts/12,16,recrc|Cmax
18	[KUN 06]	CHSP | 1, 10, 8 / / ass | / 12| (Tmin, WasteMin)
19	[LAM 96a]	DHSP|2,1,synchro,(mh,17,1),(mh,17,2)//diss|50/4,25,recrc|Other
20	[LAM 96b]	DHSP|2,12,1//diss|/nps,14|Other
21	[LEI 89a]	CHSP|1,12,ct//|/14|Tmin
22	[LEI 91]	CHSP | 2,25,ct / / diss |/ 27| Tmin
23	[LEI 93a]	CHSP|5//diss|/7|Tmin
24	[LEI 93b]	CHSP|mh,12,1//diss|/14|mhmin
25	[LEI 94]	CHSP|12//diss|/14|Tmin
26	[LIM 97]	CHSP|12//load-unload|/14|Tmin
27	[LIU 02]	CHSP | 1, 8, 5 / / load-unload | / 11, recrc| Tmin
28	[MAN 94b]	CHSP|mh,mt,ct/circ/|/nop,clean,recrc|Tmin
29	[MAN 00]	CHSP|3,17,10//load-unload|/20,clean,recrc|Tmin
30	[MAN 06]	CHSP | mh, 12, 1 / / ass | / 14| (Tmin, mhmin)
31	[MAN 08]	CHSP | mh, 14, 1 / circ/ ass | / 16| (Tmin, mhmin)
32	[MAT 00]	CHSP|8//|/nop|Tmin
33	[MAT 06]	CHSP | mt / / diss | / 2, mt+2| Tmin
34	[PAU 07]	DHSP | 18 / / diss | 40 / 4, 12| Cmax
35	[PHI 76]	CHSP | 12/ / ass | / 14, recrc| Tmin
36	[PTU 95]	CHSP|50/nc/|nparts/nps,nop|Cmax
37	[RIE 02]	CHSP|4,13,ct//|/nop,recrc|Tmin
38	[ROS 99]	PHSP|3,33,1/ret/|15/9,30,clean,recrc|Cmax
39	[SHA 88]	CHSP |1,15,6/nc,circ/|/17,clean,recrc|Tmin
40	[SON 95]	CHSP|10//diss|/12|Other

41	[SPA 99]	DHSP|1,12,3//|/14|Cmax
42	[SUB 06]	CHSP | 12/ / ass | / 14| (Tmin, WasteMin, Others)
43	[THE 90]	RHSP|3,20,1/nc/|nparts/3,nop| ThroughputMax
44	[VAR 96]	PHSP|mh,mt,1//ass|/2,nop, recrc|TransMin
45	[VAR 97]	CHSP|4,30,ct//ass|/30,recrc| Tmin
46	[VAR 00]	CHSP|mt//load-unload| /nps, nop| Tmin
47	[XU 04]	CHSP | 1, 15, 8 / / ass | / 17| (Tmin, WasteMin)
48	[YAN 01]	CHSP | mh, mt, 1 / / diss | / mt+2| Tmin
49	[YIH 93]	RHSP|5//diss|nparts/nps, 7| ThroughputMax
50	[YIH 94]	DHSP|5//diss|100/100, 7|Cmax
51	[ZHO 03]	CHSP | 1, 10, 5 / / ass | / 12| Tmin

* SHSP = PHSP with transportation times as random variables

This page intentionally left blank

Chapter 9

Shop Scheduling with Multiple Resources

9.1. Introduction

In multi-resource shop scheduling, the idea is to integrate into modeling a group of resources available for job completion. There are two types of resources. Resources unable to execute two simultaneous jobs are called disjunctive resources, for example, a machine-resource in a shop is often disjunctive. When there are several copies of the disjunctive resource in the workshop, we then speak of a shop containing several parallel resources. We distinguish the case of identical, proportional or different resources according to their performance. Resources which can execute several simultaneous jobs are called cumulative resources, for example, a team of several men can be considered a cumulative resource. The accumulation of demands on a cumulative resource cannot exceed total availability of the resource at any moment. However, whether the resources are disjunctive and made up of multiple copies or cumulative, the scheduling problem to be solved is NP-hard to its fullest extent. The goal of this chapter is to describe exact and heuristic algorithms which are intended for managing conflicts during resource usage by minimizing the makespan, i.e. the total execution time.

Next, we observe two types of scheduling problems. In section 9.2, we consider problems where circulation in the shop is the same for all jobs, and where resources are grouped by stages. We speak of hybrid flow shop, or of flow shop with duplicated machines or multiprocessor flow shop. These different names illustrate the wealth of literature in the field. Jobs circulate in the shop in the same order, and

---

Chapter written by Jean-Charles BILLAUT, Jacques CARLIER, Emmanuel NÉRON and Antoine OLIVER.

each job must visit each stage. We presume here that at least one stage contains more than one resource. In section 9.3, we discuss the most generalized multiple resource scheduling problems, where each job is made up of a partially sequenced set of operations inherent to the job, and where each operation requires one or more disjunctive or cumulative resource, in variable quantities. This is called RCPSP or the Resource Constrained Project Scheduling Problem. The hybrid flow shop is obviously a specific RCPSP, and as we will demonstrate several ideas developed for the flow shop can be applied to the RCPSP.

9.2. Hybrid flow shop scheduling problem

In practice, we find many production shops organized in "flows", i.e. shops where jobs move through machines in the same sequence. In this case, each job visits each machine in the shop and the circulation sequence on the different machines is the same for all jobs [ESQ 99, BRU 07]. We also encounter another type of shop, also organized in flows, but which presents an additional problem because machines are made up of multiple copies and grouped into "stages". In this type of shop, each job visits each stage and the circulation sequence for stages is the same for each job. The idea is to assign operations to machines at each stage. The type of shop, called hybrid flow shop or "flow shop with multiple machines", comes with a double problem: determining for each operation a start time and assigning this operation to a machine.

A hybrid flow shop example with four stages and three jobs is illustrated in Figure 9.1. Each operation *(i, j)* is referenced by a job *i* index and a stage *j* index. A processing time denoted as $p_{i,j}$ is associated with each operation, and a number of available machines denoted as m_j are associated with each stage. Two fictitious project start *(S*)* and end *(E*)* tasks are also added. The Gantt chart (Figure 9.1) represents a feasible solution for this problem.

This definition will be illustrated in section 9.2.1 with the help of production cases which can be modeled by a hybrid flow shop. Later, in section 9.2.2, a brief state of the art will be presented giving an overview of the literature in the field. In section 9.2.3, we propose a mathematical programming model of a hybrid flow shop scheduling problem including the notation used. In section 9.2.4, basic heuristic methods for easily building a heuristic solution of a hybrid flow shop problem are exposed. Section 9.2.5 is dedicated to the description of an exact method called the Branch and Bound Procedure (BBP). Finally, a generalization of the hybrid flow shop is shown in section 9.2.6.

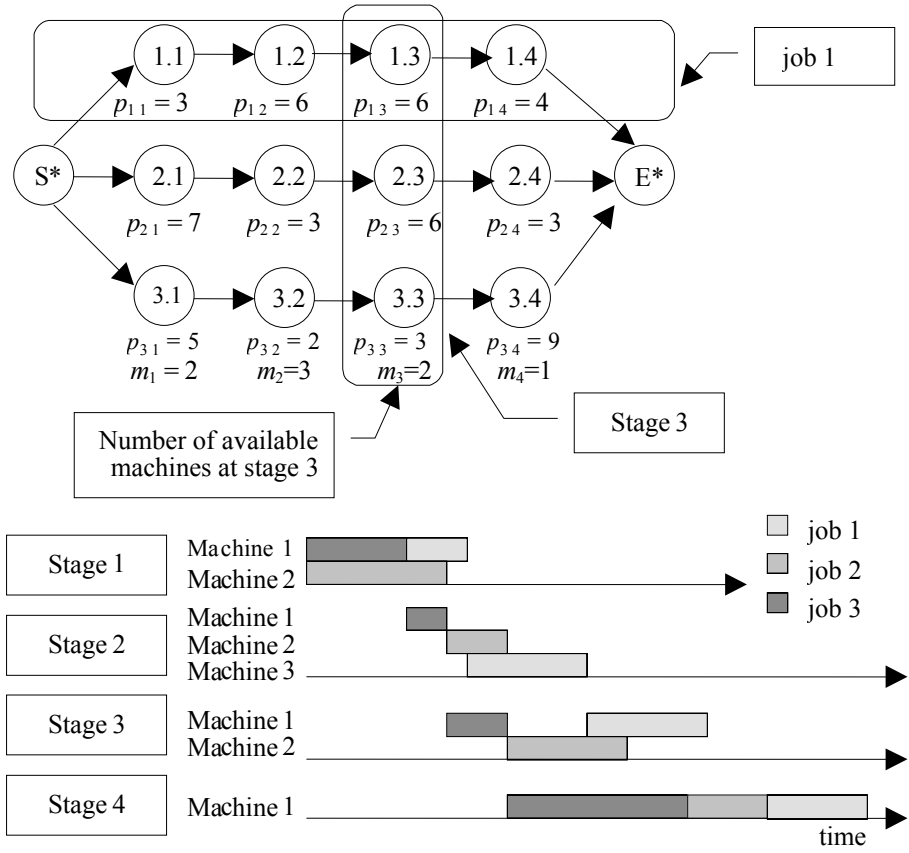


Figure 9.1. *A feasible solution for hybrid flow shop with 3 jobs and 4 stages*

9.2.1. *A few manufacturing cases*

Modeling of a shop scheduling problem by a hybrid flow shop scheduling problem is natural in many production shops. However, even outside of the production context, certain scheduling problems are hybrid flow shop type problems when there is an assignment problem to grouped resources, and entities to be scheduled follow the same flow [RUI 08]. Some cases which come directly from the manufacturing world are described below.

9.2.1.1. *Floor covering production [VIG 99]*

The first phase of floor covering production is raw weaving. In order to do this, numerous looms enable parallel production of different types of carpets (according to their variety: thickness, color, pattern, density, etc.). These machines constitute the first step of the hybrid flow shop. At this stage, it is possible to split the job into sub-lots and to execute it simultaneously on several machines. We speak of authorized splitting. The gray goods are obtained in rolls. The rolls must then be dyed. In order to do this, there are machines which continuously turn and dye the carpets. Clearly, changing dyes must be organized in order to avoid carpet waste; the wasted carpet over useful carpet ratio must be close to zero. In the shop, carpet dyeing machines can measure up to 20 meters long and weigh several tons. Since they are very expensive, there are not many copies of these machines in a shop. They can be considered as critical resources and they vary; some can be dedicated to uniform coloring while others can print color patterns. They constitute the second stage of the hybrid flow shop. The dyed carpets are then dried, latexed and packaged for delivery. This is the third stage. This shop can be illustrated by a specific hybrid flow shop containing three stages.

9.2.1.2. *Glass-bottle industry [RIC 98]*

The production process of glass bottles is semi-continuous. The continuous part is made up of a series of parallel tanks which distribute fusion glass to molding machines. Molding machines are of varied types and varied outputs, enabling the production of different bottles depending on the mold available. At a given moment, however, machines connected to the same tank can only produce bottles of the same color. The bottles then have cold and hot operations to finalize the product. This production system can be represented by a two stage hybrid flow shop with a few specific constraints. Since dye changes must be carried out at the same time for all machines, two criteria must be considered: maximizing a profit and minimizing the largest gap in relation to this dye change date.

9.2.1.3. *Production of pants [ELM 97, DES 98, WON 01]*

The context presented here is found in the production of clothes in general, as well as in other production systems where products are grouped into lots. The pant production process is broken down into several phases: cutting the fabric, sewing and stitching and installing the zippers. Each of these operations is carried out by a team of seamstresses specialized in the corresponding type of operation. Pants are grouped in packets of identical size and we presume that seamstresses do not have the same level of productivity. We can thus consider that products to be manufactured are all identical and that operation time for a product only depends on the operations and the “resource” used for this operation. This problem can be modeled by a particular hybrid flow shop with resources that have different ratings

at each stage and identical products to schedule. Sub-problems at each stage are more complex than in the case where machines are identical, but jobs are identical, which simplifies the problem. The result of the study is that from three stages on, this problem is very difficult to resolve.

9.2.1.4. *Wood treatment shops [RIA 98]*

A shop produces different basic products which will subsequently be incorporated into furniture, for example tables, chairs, sofas, etc. Each product to be manufactured must go through three consecutive operations: wood cutting, drilling and varnishing. The shop only has one machine to cut wood. The cut parts are then treated in another part of the shop made up of two machines dedicated to products depending on their size. Finally, the last operation which is meant to give the wood its final look can only happen on one machine. This shop was modeled as a particular three stage hybrid flow shop insofar as the allocation of products to machines does not have to be determined, because it depends on product size.

9.2.2. *State of the art survey*

Scheduling hybrid flow shop problems have largely been studied since the 1970s. We can find a detailed state of the art for these problems in [VIG 97, VIG 99]. The first studies focused on two stage shops as well as shops containing an undetermined number of stages. They provide heuristics as well as exact methods. Gupta [GUP 88] shows that the two stage problem, whose objective is minimization of the latest end date, is strongly NP-hard, as soon as a stage has more than one machine. The result is that all hybrid flow shop problems, except for a few specific cases, are also strongly NP-hard.

Concerning two-stage problems, many heuristics methods are based on Johnson's algorithm [JOH 54] which polynomially solves the two-machine flow shop problem with makespan minimization. This algorithm makes it possible to choose a sequence of jobs, but gives no information on the allocation of jobs to machines. Certain authors propose a complement to allocation rules [GUP 88, SRI 89, DEA 91, LEE 93, LEE 94, SRI 89], while others do not [SHE 72, BUT 73]. For problems with other criteria and specific constraints, the authors proposed inventive heuristics [HAO 97, GUP 98, RIA 98, TKI 00, CHA 04, HE 07, LOW 08] and exact methods dedicated to the problem [GUP 91, GUP 94, LI 97, GUP 95, LEE 04].

Concerning multi-stage problems, numerous heuristic methods were developed. Often, they are inspired by heuristics used for the flow shop enabling the determination of a job sequence. As with the two stage case, we must then add rules for the allocation of jobs to machines [LEE 94, GUI 96]. The stochastic methods

(see Chapter 3) were largely used. For example, Nowicki and Smutnicki [NOW 98] present a Tabu algorithm with very good results. Evolutionist algorithms like genetic algorithms (see Chapter 4) were also proposed to solve these problems [POR 98, MOR 05]. In addition, several BBPs were described to solve the basic problem (no particular constraint and minimization of the latest end date) in an exact way. Brah and Hunsucker [BRA 91] have adapted a BBP from Bratley, Florian and Robillard [BRA 75]. As it turns out, this method is not very powerful despite a few proposed improvements [POR 98, DUP 98]. Since then, numerous other BBPs were developed and offer very good results [VAN 94, CAR 97, CAR 98, BRO 98, DES 98, NER 98, MOU 00].

9.2.3. *Notation and mathematical model*

T is the set of n jobs, *K* is the number of stages and m_j is the number of machines at stage *j*. M_j is the set of machines constituting stage *j*. We denote *(i, j)* the operation of job *i* processed at stage *j* and $p_{i,j}$ is its duration. In short $p_j = p_{i,j}, \forall\, i = 1...n,$ is the operation processing time vector at stage *j*. In the basic problem, we consider that the machines are all available at date 0, that jobs can all start at date 0, and that the objective is to minimize the maximum completion time. The objective of the following linear programming model is to formalize the problem and not to propose a resolution method. We denote by $t_{i,j}$, the start time of *(i, j)*; $x_{i,j,k}$ a binary variable equal to 1 if *(i, j)* is performed on machine *k* and 0 otherwise; and by $y_{i,i',j}$ a binary variable equal to 1 if job *i* precedes job *i'* at stage *j*, 0 otherwise. HV is an arbitrarily large number. A linear programming model is:

minimize $C_{\max}$

subject to:

$$\sum_{k \in M_j} x_{i,j,k} = 1, \ \forall i, 1 \le i \le n, \forall j, 1 \le j \le K \qquad [9.1]$$

$$C_{\max} \ge t_{i,K} + p_{i,K}, \forall i, 1 \le i \le n \qquad [9.2]$$

$$t_{i,j+1} \ge t_{i,j} + p_{i,j}, \forall i, 1 \le i \le n, \forall j, 1 \le K\text{-}1 \qquad [9.3]$$

$$t_{i'j} \ge t_{i,j} + p_{i,j} - HV.(1 - y_{i,i',j}) - HV. \sum_{a \in M_j, a \ne k} (x_{i,j,a} + x_{i',j,a}), \qquad [9.4]$$

$$\forall i, 1 \le i \le n, \forall i', 1 \le i' \le n, \forall j, 1 \le j \le K, \forall k, k \in M_j$$

$$t_{i,j} \geq t_{i',j} + p_{i',j} - HV.y_{i,i',j} - HV. \sum_{a \in M_j, a \neq k} (x_{i,j,a} + x_{i',j,a}), \quad [9.5]$$

$$\forall i, 1 \leq i \leq n, \forall i', 1 \leq i' \leq n, \forall j, 1 \leq j \leq K, \forall k, k \in M_j$$

Constraint [9.1] makes sure that operation *(i, j)* is assigned to one machine, constraints [9.2] define the $C_{\max}$, [9.3] translate the routing constraints, and constraints [9.4] and [9.5] translate the succession of operations on machines. This mathematical model can be theoretically processed by a commercial software package, but the problems which can be solved this way are in reality quite small.

9.2.4. *Heuristic canonical methods*

In order to address a hybrid flow shop problem and provide a first resolution algorithm, we should not bypass the most traditional methods. These methods consist of determining a sequenced list of jobs and then to consider each job, in the sequence of the list one after the other, and to choose a resource to execute it by using an allocation rule. Several algorithms make it possible to determine a job list:

– for a 2-stage problem, we can apply the Johnson algorithm, or define an index and sort jobs according to their increasing index values, or even use a traditional priority rule such as SPT (Shortest Processing Time first) or EDD (Earliest Due Date first) by considering a specific stage or by adding times on stages. In any case, the result is a sorted job list;

– for an n-stage problem, we can apply a traditional heuristic for the flow shop problem such as CDS [CAM 70], Townsend [TOW 77] or NEH [NAW 83] all providing a job sequence.

Numerous rules then make it possible to assign the jobs to machines. We can mention for example:

– FAM (First Available Machine) consisting of assigning the job to the first available machine. This is the most widely used rule;

– ECT (Earliest Completion Time) if machines are not identical, i.e. jobs have processing times which depend on the machine used; we can assign the job to the machine making it possible to complete earlier;

– LSM (Latest Start Machine), which consists of choosing the latest available machine among available machines when the operation is ready (at a given date in the first stage or completed at the previous stage), or the earliest available machine if no machine is available when the operation is ready. This rule tends to minimize the idle time of used machines and to effectively only use those which are necessary;

– assign jobs in the machine numbering sequence, if they are identical. This allocation minimizes the sum of completion times in a parallel machine problem, if the jobs are classed beforehand according to the SPT rule.

An algorithm to determine a job list and an allocation rule make it possible to build a first heuristic for the hybrid flow shop problem. Concerning quality of results, this type of method is much less powerful on average than stochastic algorithms of simulated annealing or Tabu search type, or even genetic algorithms. However, it has the advantage of being simple to understand and implement. Furthermore, such a method can be used to obtain an initial solution that can be improved by a stochastic algorithm [JIN 06, LAH 07].

This heuristic can be summarized as follows. At the first stage of the hybrid flow shop, we take the jobs in the list order and we assign them by following the established rule. There are two possibilities at a subsequent stage. We either consider the jobs again according to the list, as was defined for the first stage, or we build a new list in which jobs are classified in increasing order of their due date in the previous stage. We then apply the allocation rule by taking into consideration availability dates for allocating all jobs and we start again at the following stage. These two different processes, which are almost never mentioned in other works, lead to different solutions as the following example shows.

EXAMPLE.– Consider a two stage hybrid flow shop with two machines at each stage and three jobs to schedule. The objective is to minimize the maximum completion time. Job processing times are equal to $p_1 = (2, 15, 6)$ for the first stage, and $p_2 = (11, 4, 9)$ for the second stage. The Johnson algorithm gives a J = (1, 3, 2) sequence, the SPT rule applied to the first stage gives a SPT1 = (1, 3, 2) sequence, and applied to the second stage SPT2 = (2, 3, 1). For example we will use sequence SPT2. We choose to use the FAM allocation rule in the first and second stage. The Gantt chart in Figure 9.2 represents the allocation result for the first stage.

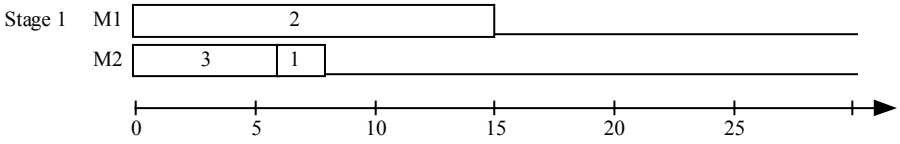


Figure 9.2. *Gantt chart for the first stage*

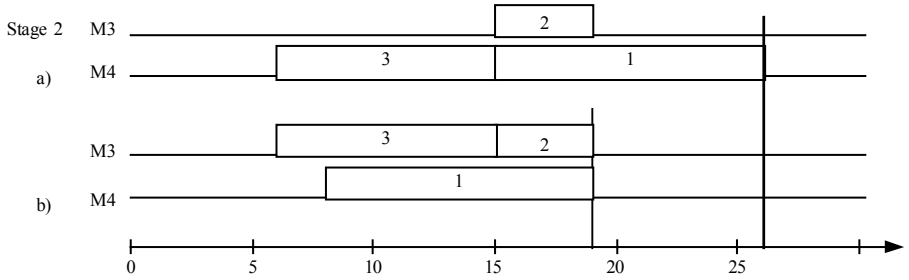


Figure 9.3. *Gantt charts of different solutions at the second stage*

For the second stage, two cases must be considered: a) we keep the list given by SPT2 rule, or b) we build a new list according to completion times of jobs in the first stage. In the second case, we obtain list L = (3, 1, 2). Corresponding solutions are represented in Figure 9.3. We observe that the solutions are not the same.

The authors very often place themselves in case b) without being explicit, which minimizes the importance of the sequence used in the first stage. We can also note that the case may sometimes generate non-active schedules, which is not desirable when the criteria to be optimized are regular.

9.2.5. *An exact method*

In addition to the heuristic methods, for which the challenge consists of finding solutions which get closer to the optimal solution, several exact methods have been developed, and their challenge consists of optimally solving increasingly large problems. Among the exact methods very often used in scheduling literature, we find BBPs. The hybrid flow shop BBP presented here involves traditional hybrid flow shop, i.e. without a specific constraint, with K stages and with the goal of minimizing the maximum completion time (C_{max}). This method, called the "interval method" [CAR 84, CAR 91, VAN 94, PER 95, MOU 99, OLI 99], consists of dividing time intervals associated with operations as much as possible. The branching scheme as well as associated lower bounds are described below. The method presented in the previous section provides upper bounds.

We associate with each operation *(i, j)* an earliest start time $r_{i,j}$, a latency duration $q_{i,j}$, a due date or latest completion time $d_{i,j}$ and a slack $s_{i,j}$ defined in the following way (UB designates an upper bound of the C_{max} value):

$$r_{i,j} = \sum_{k=1}^{j-1} p_{i,k} \ , \ q_{i,j} = \sum_{k=j+1}^{K} p_{i,k} \ , \ d_{i,j} = UB - q_{i,j} \ , \ s_{i,j} = d_{i,j} - r_{i,j} - p_{i,j}$$

A BBP node corresponds to a specific operation to be chosen and a series of dates:

– an operation number *(a, b)* (job *a*, stage *b*);

– the list of all earliest start times;

– the list of all latest completion times;

– the list of all latency durations;

– the value of a lower bound for the evaluation of a node.

9.2.5.1. *Branching scheme*

The principle of the *branching scheme* of this BBP consists of creating, in each *P* node, two sub-problems by delaying the earliest operation start time of *(a, b)* for the Left Son *(LS)* node, and by advancing the latest completion time of *(a, b)* for the Right Son *(RS)* node. By reducing the time intervals associated with an operation, we constrain the problem, which makes it possible to quickly converge toward the optimal solution with the help of lower and upper bounds. The BBP algorithm is presented below (Algorithm 9.1).

The update of the earliest start and latest completion times of son nodes of an ordinary *P* node is carried out in the following way:

– for the *LS* node, dates associated with operations are the same as for the *P* node, except for the earliest start times for operations of job *a*, which are updated in the following manner:

$$r_{a,j}(LS) = r_{a,j}(P),\ 1 \le j \le b\text{-}1, \text{ and } r_{a,b}(LS) = r_{a,b}(P) + \left\lceil \frac{s_{a,b}(P)}{2} \right\rceil$$

and $r_{a,j}(LS) = max[r_{a,j}(P), r_{a,j-1}(LS) + p_{a,j-1}]$ for j from $b+1$ to K

– for the *RS* node, dates associated with operations are the same as for the *P* node, except for the latest completion times for operations of job *a*, which are updated in the following manner:

$$d_{a,,j}(RS) = d_{a,j}(P),\ b+1 \le j \le K \text{ and } d_{a,b}(RS) = d_{a,b}(P) - \left\lfloor \frac{s_{a,b}(P)}{2} \right\rfloor - 1,$$

$$d_{a,,j}(RS) = min[d_{a,,j}(P),\ d_{a,j+1}(RS) - p_{a,j+1}] \text{ for } j \text{ from } b-1 \text{ to } 1$$

9.2.5.2. *Lower bounds [CAR 84, SAN 95]*

For node *N*, consider stage *j*. It is possible to calculate the following lower bound $LB_j(T,N)$ from earliest operation start times and latency durations.

$$LB_j(T,N) = \left\lceil \frac{r_{[1],j}(N) + \cdots + r_{[m_j],j}(N) + q_{[1],j}(N) + \cdots + q_{[m_j],j}(N) + \sum_{i \in T} p_{i,j}}{m_j} \right\rceil$$

where $r_{[1],j}(N)$, ..., $r_{[mj],j}(N)$ are the m_j smallest earliest start times and $q_{[1],j}(N)$, ..., $q_{[mj],j}(N)$, are the m_j smallest latency durations for *N*.

EXAMPLE.– Consider the data from the previous example. At stage 1, suppose that a heuristic algorithm gives *BestUB = 21* as upper bound. The initial date calculation gives the following results.

Root node *			
$r_{i,j}$	i = 1	i = 2	i = 3
j = 1	0	0	0
j = 2	2	15	6
$q_{i,j}$	i = 1	i = 2	i = 3
j = 1	11	4	9
j = 2	0	0	0
$d_{i,j}$	i = 1	i = 2	i = 3
j = 1	10	17	12
j = 2	21	21	21

Table 9.1. *Root BBP node*

// initialization
Calculate an upper bound BestUB with the help of a heuristic,
for *each operation (i, j)*
  calculate $r_{ij}(*)$, $q_{ij}(*)$, $d_{ij}(*)$,
end for
for *each stage j,*
  calculate the lower bound $LB_j(T, *)$.
end for
$LB^* \leftarrow max_{1 \leq j \leq K} LB(T,*)$.
Create the root node (*).
While *there remains an unexplored node*
  // Step 2 choice of a node to explore
  Choose node N such that $LB(N) = min_p LB(P)$.
  if $LB(N) = BestUB$ ***then*** *END, Display BestUB and the schedule corresponding to N*
  otherwise
  // Step 3 update of earliest start times and latency durations
  for *any operation (i, j)*
    $q_{i,j}(N) \leftarrow max\{q_{i,j}(N), LB(N) - d_{i,j}(N)\}$
    $d_{i,j}(N) \leftarrow min\{d_{i,j}(N), BestUB - q_{i,j}(N)\}$
  end for
  launch the calculation of a heuristic for updating BestUB.
  // Step 4 Choice of stage
  for *any stage j*
    calculate $LB_j(T, N)$
  end for
  $j^* \leftarrow arg\{ max_{1 \leq j \leq K} LB_j(T, N)\}$
  // Step 5 update of earliest start times and latency durations
  determine E_r *the set of* m_{j*} *jobs with smallest earliest start times*
  determine E_q *the set of* m_{j*} *jobs with smallest latency durations*
  for *each job i of* E_r
    calculate new earliest start times $nr_{i,j*}(N) \leftarrow r_{i,j*}(N) + \lceil s_{i,j*}/2 \rceil$.
    create a fictitious node LS(i) and calculate new lower bounds $LBj^*(T, LS(i))$.
  end for
  for *each job i of* E_q
    calculate new latest completion times $nr_{i,j*}(N) \leftarrow r_{i,j*}(N) + \lfloor s_{i,j*}/2 \rfloor$
    create a fictitious node RS(i) and calculate new latency times and then new lower bounds $LBj^*(T, RS(i))$.
  end for
  Or job $i^* \in E_r \cup E_q$ *such that the difference between the associated lower bound and the lower bound of N be maximal.*
  // Step 6 Node creation
  Lower bound update
  Add to list of nodes LS(i) and RS(i*)*
  end if
end while

Algorithm 9.1. *BBP algorithm*

At step 1, lower bound calculation for each stage gives: $LB_1(T,*) = ((0 + 0) + (4 + 9) + 23) / 2 = 18$, for the first stage and $LB_2(T,*) = ((2 + 6) + (0 + 0) + 24) / 2 = 16$, for the second. We have $LB(*) = 18$. At step 2, the chosen node is the root node * (single node). At step 3, we update latency durations and latest completion times. Calculation shows that their values do not change, neither does the upper bound. At step 4, lower bound calculation leads to the same result, we then select the first stage: $j^* = 1$. At step 5, we then obtain E_r={1, 2} and E_q={2, 3}. For each E_r job, we calculate new earliest start times and for each E_q job, new latest completion times.

For E_r we obtain the two following tables.

Job 1 ($s_{11} = 10 – 0 – 2 = 8$)			
Fictitious node *LS(1)*			
$nr_{i,j}$	i = 1	i = 2	i = 3
j = 1	4	0	**0**
j = 2	6	15	**6**
$q_{i,j}$	i = 1	i = 2	i = 3
j = 1	11	**4**	9
j = 2	0	0	0
$d_{i,j}$	i = 1	i = 2	i = 3
j = 1	10	**17**	12
j = 2	21	21	21

Job 2 ($s_{21} = 17 – 0 – 15 = 2$)			
Fictitious node *LS(2)*			
$nr_{i,j}$	i = 1	i = 2	i = 3
j = 1	0	1	0
j = 2	2	16	6
$q_{i,j}$	i = 1	i = 2	i = 3
j = 1	11	4	**9**
j = 2	0	0	0
$d_{i,j}$	i = 1	i = 2	i = 3
j = 1	10	17	**12**
j = 2	21	21	21

Table 9.2. *Calculation of two fictitious nodes "on the left" for the BBP*

We find: $LB_1(T, LS(1)) = ((0 + 0) + (4 + 9) + 23)/2 = 18$ and $LB_1(T,LS(2)) = 18$ also. For E_q, we find the following.

Job 2 (s_{21} = 17 – 0 – 15 = 2)			
Fictitious node *RS(2)*			
$r_{i,j}$	i = 1	i = 2	i = 3
j = 1	0	0	**0**
j = 2	2	15	**6**
$q_{i,j}$	i = 1	i = 2	i = 3
j = 1	11	4	9
j = 2	0	0	0
$nd_{i,j}$	i = 1	i = 2	i = 3
j = 1	10	15	12
j = 2	21	21	21

Job 3 (s_{31} = 12 – 0 – 6 = 6)			
Fictitious node *RS(3)*			
$r_{i,j}$	i = 1	i = 2	i = 3
j = 1	0	0	0
j = 2	2	15	6
$q_{i,j}$	i = 1	i = 2	i = 3
j = 1	11	4	11
j = 2	0	0	0
$nd_{i,j}$	i = 1	i = 2	i = 3
j = 1	10	17	**8**
j = 2	21	21	21

Table 9.3. *Calculation of two fictitious nodes "on the right" for the BBP*

We find: $LB_1(T,RS(2))$ = ((0 + 0) + (4 + 9) + 23) /2 = 18 and $LB_1(T,RS(3))$ = ((0 + 0) + (4 + 11) + 23) /2 = 19. The selected job is the one with the highest difference with the old lower bound, in this case job 3. The connecting operation is operation (3,1). At step 6, we add the following two nodes to the node list.

Node 1 = *LS(3)*			
$r_{i,j}$	i = 1	i = 2	i = 3
j = 1	0	0	3
j = 2	2	15	9
$q_{i,j}$	i = 1	i = 2	i = 3
j = 1	11	4	9
j = 2	0	0	0
$d_{i,j}$	i = 1	i = 2	i = 3
j = 1	10	17	12
j = 2	21	21	21

Node 2 = *RS(3)*			
$r_{i,j}$	i = 1	i = 2	i = 3
j = 1	0	0	0
j = 2	2	15	6
$q_{i,j}$	i = 1	i = 2	i = 3
j =1	11	4	**11**
j = 2	0	0	0
$d_{i,j}$	i = 1	i = 2	i = 3
j = 1	10	17	**8**
j = 2	21	21	21

Table 9.4. *Creation of two real nodes for the BBP*

We again calculate the lower bounds associated with these two new nodes. We find 18 for *LS(3)* and 19 for *RS(3)* and we go back to step 2.

Finally, C_{max} value for the optimal solution is equal to 19. The values obtained for earliest start times and latest completion times lead to the solution illustrated in the Gantt chart in Figure 9.4.

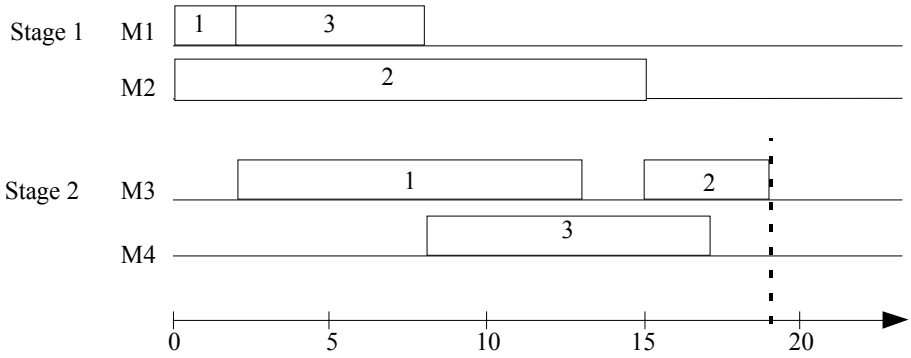


Figure 9.4. *Gantt chart of the resulting BBP optimal solution*

We can observe that the solution found by the heuristic in this small example is optimal for case b) (see Figure 9.3).

9.2.6. *Extensions of the traditional hybrid flow shop problem*

A first traditional hybrid flow shop problem extension consists of considering that machines of one stage do not have the same outputs (see pants manufacturing example in section 9.2.1). In this case, we consider that the operation processing time is based on the performing machine. Clearly, associated problems are more difficult to solve than in the case where machines are identical, but they more closely resemble manufacturing reality.

We find another extension of this problem in certain manufacturing shops containing machines with their adapted tools. Each machine contains a certain number of tools and consequently can only execute certain types of operations. We then speak of *multi-purpose machines* [JUR 92, BRU 97]. At each stage, the tools are considered to be distributed between the machines. Each operation requiring a specific tool for its execution can only be executed on a subset of machines in that stage. In a way, this problem consists of restricting the number of possible allocations to a stage. The most common case consists of presuming that all shop machines are different and that each machine has specific tools. In this case, each operation can only be executed on a series of machines inherent to the operation

with time depending on the machine chosen. In reality, we find several applications that can be illustrated that way (see Chapter 12).

Finally, a natural hybrid flow shop extension consists of considering that an operation may require several types of machine for its execution, and for each machine type, a certain number of machines. If we consider that machines are no longer organized in stages, then we obtain a multi-resource (each operation requires more than one type of resource) cumulative (each operation may require more than one resource unit per resource type) problem. In addition, if precedence constraints between operations are not chains (job constraints), but given by a graph of precedence, then we obtain a problem called "resource constrained project scheduling".

9.3. RCPSP: presentation and state of the art

The Resource Constrained Project Scheduling Problem (RCPSP) is widely studied in other works [BRU 99, DEM 02, WEG 05, ART 08]. It consists of scheduling a given number of tasks over one or more limited capacity resources. Each task is defined by a processing time, consumption of each resource, and a series of tasks called predecessors, i.e. a task cannot start before the end of all its predecessors. The goal is then to find feasible schedules, i.e. task start times which satisfy resource constraints as well as precedence constraints, and that optimize given criteria, such as the project completion time for example. Many works have shown that this problem can be used to solve real life applications, and several extensions have been proposed. See [BRU 99, OZD 95, KOL 97, DEM 02, WEG 05, ART 08] for a description of solving methods which consider variations of the RCPSP.

Many heuristic methods have been proposed for solving the RCPSP: priority rule-based methods [KLE 00], neighborhood and large neighborhood search [PIN 94, BOU 03, DEB 06, FLE 04, GOD 05, KOC 03, PAL 04, VAL 03], population-based method [MER 02, VAL 04], activity-insertion-based methods [ART 00, ART 03], etc. Experimental evaluation of some of these methods has been recently presented [KOL 06].

Authors have also proposed efficient lower bounds [MING 98, BRU 98a, BRU 00, CAR 03, CAR 07, DAM 05, DEM 05].

In this chapter we present the traditional form of the problem (section 9.3.1), along with the main exact resolution methods (section 9.3.2).

9.3.1. *A simple model including shop problems*

The RCPSP generalizes traditional scheduling problems such as job shop or flow shop. This time, a task may have a certain number of predecessors and/or successors. However, the graph associated with precedence constraints between tasks should not have a directed cycle in order for the problem to accept a solution. In addition, tasks are no longer executed on a machine, but require one or more cumulative resources. This problem is characterized by:

– $X = \{1, 2, \ldots, n\}$: all tasks. Tasks 1 and *n* are the fictitious project start and end tasks;

– $U = \{(i, j), \ldots, (h,l)\}$: all precedence constraints between tasks; $(i,j) \in U$ if j cannot start before the end of i;

– p_i: task i processing time, also called duration;

– $R = \{1, 2, \ldots, k\}$: all resources used by tasks;

– A_k: availability of resource k, also called resource capacity;

– $a_{i,k}$: resource k quantity required by task I, also called resource requirement, or resource consumption;

– r_i: task i release date initialized at l(1, i), where l(i,j) is the longest path in the precedence graph between task i and task j;

– q_i: task i tail initialized at $l(i, n) - p_i$;

– d_i: deadline of task i defined in the case where project completion date is set to D: $d_i = D - q_i$.

The notations presented above correspond to the most traditional version of the RCPSP. The goal is then to find the minimal length schedule which does not violate precedence constraints or resource constraints. More general models were presented in the literature to take into account more complex systems. A problem typology and notation were proposed in [HER 98b] to generalize the traditional typology.

It is important to note that this problem is NP-hard [GAR 79]. An example of the RCPSP is presented below.

EXAMPLE.– We consider n = 9 tasks with respective processing time, 0, 2, 2, 4, 1, 3, 2, 3, 0, and k = 3 resources with availability $A_1 = 3$, $A_2 = 3$ and $A_3 = 2$. Figure 9.5 shows precedence constraints between tasks, and the Gantt chart shows a feasible solution of duration 12.

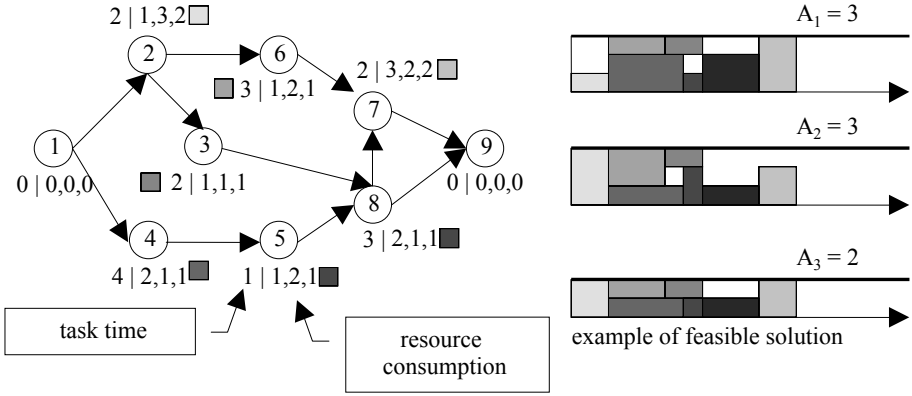


Figure 9.5. *Example of the RCPSP*

9.3.2. *Main exact methods for the RCPSP*

Since the RCPSP is NP-hard, tree search-based methods, or Branch-and-Bound (B&B), are preferred tools for its exact resolution. We intend to present a few exact reference methods. A tree search-based method has already been proposed in section 9.2.5. We remind our readers that a B&B is based on a few essential mechanisms:

– the branching scheme, for building the search tree;

– lower bound, which provides a lower bound of the best solution associated with the current node;

– dominance rules, establishing the dominance of one node over another;

– adjustment rules for deriving information from decisions taken during previous steps of the tree search.

9.3.2.1. *Traditional branching schemes*

This section focuses on the presentation of a few branching schemes commonly encountered in the literature.

Chronological schemes

When a chronological scheme is used, a node (P) in the search tree corresponds to a series of scheduled tasks SC(P), i.e. a group of tasks for which the starting time is set. The eligible series of tasks EL(P), i.e. the group of tasks with scheduled predecessors, is determined. A task $i_p \in EL(P)$ is chosen. The time t_p is determined as

the earliest time point for placing i_p, without violating constraints of precedence or resource constraints, and which is larger than or equal to the moment of placement of the last placed task.

Most authors, including Sprecher [SPR 96], calculate in each node the series of eligible tasks as well as their earliest start times. If backtracking occurs, another EL(P) task is chosen to be scheduled. If all EL(P) tasks have been tested, we go back up in the search tree. In the tree thus generated, a node (P) has as many sons as there are tasks in EL(P). Baptiste and Le Pape [BAP 97] use a slightly different chronological branching scheme: for the i_p chosen, either i_p is scheduled before all eligible tasks, or a constraint between task i_p starting time and the starting time of the EL(P) group's other tasks is set. We mandate that at least one task of this series starts before the beginning of i_p.

Shifting of a minimal series of tasks

Demeulemeester and Herroelen [DEM 92, DEM 97] use a branching scheme inspired by the one proposed by Christofides *et al.* [CHR 87], based on minimal subsets of deferred tasks to solve new resource conflicts.

Eligible subsets

The branching scheme presented in [MIN 98] is based on the extension of partial scheduling by block addition at each node in the search tree. A subset of tasks defines a block if and only if constraints of resources and precedence are not violated when all tasks in the block are executed at the same time.

Reduction of task intervals

Carlier and Latapie [CAR 91] proposed a branching scheme which is not directly based on the construction of partial schedules. Task deadlines are then presumed to be known. A set of tasks corresponds to a node (P) in the search tree. Each of these tasks has a $r_i(P)$ release date and a deadline $d_i(P)$. Task i_p is chosen. Two sons are then created: P_1, for which the release date of task i_p is modified, and P_2, for which the deadline of task i_p is modified. For more information about this method, please refer to section 9.2.5 where it was presented for the resolution of the hybrid flow shop which is a specific RCPSP case.

Disjunctions, conjunctions and parallelism

This branching scheme was proposed in [BRU 98a]. It is based on the "schedule scheme" notion (C, D, N, F) which indicates, for a set of tasks, task pairs in conjunction, pairs in disjunction, pairs which must be executed in parallel and flexible pairs, for which none of the relations mentioned is mandated. For each node, a task pair is chosen among those still flexible and a relation (disjunction or

parallelism) is set, then two nodes are created, one corresponding to fixing a disjunction and the other corresponding to fixing parallelism relation.

9.3.2.2. *Lower bounds*

A lower bound, applied to a node (P) in the arborescence, provides a lower bound for the length of the best solution which will be obtained from (P). In this way, the search tree can be limited by excluding nodes from which no strictly better solution than the best known solution can be found.

Critical path bounds

The first bounds used for the RCPSP are based on the critical G = (X, U) graph path notion. Release dates and tails are those corresponding to the node for which the evaluation is calculated.

Lower bounds for the m machine problem

m machine problems are a relaxation of the initial RCPSP. The basic idea is to simply translate resource conflicts by relaxing constraints of precedence. If S is a set of tasks and if, for any subset *s* of S, such that $|s| > m$, all s tasks cannot be executed together without violating one of the resource constraints, the *m* machine problem in which each S task requires one machine to be executed, can be introduced.

These are a few commonly used bounds for this problem finding their place in a tree search-based method for the resolution of the RCPSP. Quantity $G'(J) = (r_{i_1}+\ldots+ r_{i_m} + \Sigma_{i\in J}\ p_i+ q_{j1}+\ldots q_{jm})/m$ is the bound associated with the m machine problem [CAR 84], used by Carlier and Latapie [CAR 91], with $r_{i1},\ldots, r_{im}$ the smallest m release dates of tasks J and $q_{j1},\ldots,q_{jm}$ the smallest m tails (see section 9.2.5). This bound was improved with bounds called "subset bound" and "adjusted subset bound" [PER 95, VAN 94]. We should also mention the development of the Jackson pseudo-preemptive schedule by Carlier and Pinson [CAR 98a], for directly obtaining the subset bound.

Time bound reasoning applied to the cumulative scheduling problem

Baptiste *et al*. [BAP 99] focused on the cumulative scheduling problem (CuSP), which is a relaxation of a decisional occurrence of the RCPSP to one of its resources in which constraints of precedence are replaced by $[r_i, d_i]$ intervals, associated with each task i. Since index k of the resource is set, it will subsequently be omitted. The authors have particularly proposed a series of relaxations for this problem in order to obtain necessary existence conditions. This relaxation, called "non-interruptible", is based on the notion of energy [LAH 82, LOP 92], presented in Chapter 5 and reviewed here.

For a given *i* task, the minimum energy required by i over any interval $[t_1, t_2]$ is equal to (see section 5.3.3): $w_i^{[t_1, t_2]} = a_i$. min (max (0, $r_i + p_i - t_1$), max (0, $t_2 - d_i + p_i$), p_i, $t_2 - t_1$). Lopez *et al.* [LOP 92] have proven that, if a non-empty interval $[t_1, t_2]$ exists for which $\Sigma_i w_i^{[t_1, t_2]} > A . (t_2 - t_1)$, then no feasible schedule will be found for the CuSP. Baptiste *et al*. [BAP 99] have demonstrated that it is simply required to verify this necessary existence condition for a quadratic number of intervals. A quadratic algorithm is proposed to establish the necessary existence conditions for all relevant intervals.

A linear program based on the minimum feasible subsets

The lower bound proposed by Mingozzi *et al.* [MIN 98] is based on blocks as previously defined. A linear program involving two types of decision variables is proposed. The first variables decide if a given block is executing at a given moment. The second variables express the fact that a task *i* starts at a given moment. Brucker *et al.* [BRU 98a] use one of the intermediate problem relaxations and apply an exact resolution by column generation.

9.3.2.3. *Adjustments and rules of dominance*

As reviewed above, rules of dominance and adjustment eliminate nodes of the search tree, for which no interesting solution can be developed. These rules generally apply to the structure of the partially generated scheduling, or to the search memory, i.e. that certain information is stored during the search to establish a comparison between the current node and the ones already explored. Generally, rules based on search memory belong to a class of techniques often used in artificial intelligence: intelligent backtracking. Here, starting times for scheduled tasks are the information stored. Adjustment processes of release dates and tails, as well as other mechanisms can be introduced. They make it possible to "over-constrain" the problem by getting as much information as possible from decisions made.

Left-shift

This rule of dominance, initially proposed by Stinson *et al.* [STI 78], is based on the fact that all active and semi-active schedules, as formally defined for the RCPSP by Sprecher *et al.* [SPR 95], contain a minimum time solution. In this way, numerous methods [BEL 90, DEM 92, DEM 97, MIN 98, SPR 96], based on the development of partial schedules test if a sequenced operation can be left-shifted. In this case, the partial solution considered is not explored.

The cut-set rule

The rule of dominance called cut-set was described by Demeulemeester and Herroelen [DEM 92]. It consists of comparing the partial schedule in development to a previously developed partial schedule. Methods based on the same type of

comparison were previously proposed by Stinson *et al.* [STI 78], and Talbot and Patterson [TAL 78]. The cut-set rule actually comes down to comparing two partial schedules containing the same tasks. If, in the first one, all tasks end sooner than in the second, then all the solutions to be deduced from the first one will also be deduced from the second one. This rule has been widely used since its implementation by Demeulemeester and Herroelen [DEM 97, HER 98a, MIN 98].

Time bound adjustments

Contrary to the two previous rules which enabled the elimination of nodes when they were dominated, time bound adjustments described by Baptiste *et al.* [BAP 99], make it possible to over-constrain the problem and to adjust the intervals $[r_i, d_i]$ of tasks when it is possible, without creating another choice point in the search tree.

This technique is based on the time bound reasoning proposed by Lahrichi [LAH 82], Erschler *et al*. [ERS 91], and Lopez *et al.* [LOP 92]. These adjustments are based on the notion of mandatory task part, whether it shifts to the left ($p_i^+(t_1)$) or to the right ($p_i^-(t_2)$), in relation to an interval $[t_1, t_2]$: $p_i^+(t_1) = \max(0, p_i - \max(0, t_1 - r_i))$, $p_i^-(t_2) = \max(0, p_i - \max(0, d_i - t_2))$. The energy required for task i over interval $[t_1, t_2]$ is thus defined as: $w_i^{[t1, t2]} = a_i \,.\, \min(t_2 - t_1, p_i^+(t_1), p_i^-(t_2))$, and $W^{[t1, t2]} = \Sigma_i\, w_i^{[t1, t2]}$. Therefore, if: $\exists [t_1, t_2] / W^{[t1, t2]} - w_i^{[t1, t2]} + a_i \,.\, \min(t_2 - t_1, p_i^+(t_1)) > A \,.\, (t_2 - t_1)$, then a lower bound of the completion time of task i is: $t_2 + (1/a_i).(W^{[t1, t2]} - w_i^{[t1, t2]} + a_i \,.\, \min(t_2 - t_1, p_i^+(t_1)) - A \,.\, (t_2 - t_1))$. For this earliest end date, we can update the task's release date. The symmetric adjustment is also defined for deadline (tails).

Introduction of disjunctions

All deduction methods proposed by Brucker *et al*. are grouped under this terminology [BRU 98a]. All these methods were developed to significantly improve the performance of the branching scheme based on a representation (C, D, N, F). These methods enable the setting of disjunctions in "flexible" task pairs without introducing a choice point, i.e. without developing a node in the arborescence.

9.3.2.4. *Development of an exact method in a simple example*

We propose an explanatory example showing the main mechanisms involved in the resolution of the RCPSP. To illustrate our example, we have chosen the method proposed by Baptiste and Le Pape [BAP 97, BAP 99, NER 99]. This method is based on:

– consecutive resolutions of decision-making instances;

– the development of partial schedules according to a chronological branching scheme;

– the application of time bound reasoning as necessary existence conditions.

```
// initialization
calculate UB: best known solution time (list method);
calculate LB: a lower bound (larger value for which necessary time bound conditions
    detect no inconsistency;
// dichotomous search on the smallest eligible UB bound
while (LB < UB)
    Determine UB_test the bound to test UB_test ← (UB + LB)/2
    // initialization of the tree search method
    create the corresponding root node: fictitious start task is sequenced at t = 0;
    t ← 0,
    choose a task (e) which does not have task 0 as its predecessor
    // scan the search tree
    while (there is an unexplored node in the stack)
    remove the first node from stack // e is the task to place
    // placement of chosen task
    place (e) at t_e the first moment larger than t and r_e such that sufficient resources are
    available
    if all tasks are sequenced then
        UB ← scheduling end date
        if scheduling end date ≤ UB_test then
            UB ← UB_test
            END_UB
        end if
    end if
    t ← t_e
    for non-placed tasks r_i ← max (r_i, t_e)
    propagate updates on graph
    if inconsistency is detected (∃i/ d_i > UB_test) then END_NODE
    // application of time bound necessary conditions
    apply necessary conditions and time bound adjustments
    If inconsistency is detected then END_NODE
    propagate these adjustments in the graph
    if inconsistency is detected (d_i > UB_test) then END_NODE
    // branching function
    determine the list of tasks where predecessors are finished
    choose (e_n) one of these tasks // i.e. the one with the smallest r_i
    place on the stack of unexplored nodes the node in which at least one task will start
    before start of (e_n)
    place on stack of unexplored nodes the node in which the next placed task will be (e_n)
    end as long as // end of node
    if no solution with time smaller than or equal to UB_test was found then LB ← UB_test +
    1
end while  // end of the method
```

Algorithm 9.2. *"Chronological" BBP algorithm*

Algorithm 9.2 briefly recaps the structure of this method. In this general algorithm, we have voluntarily omitted certain aspects, i.e. the initial branch phase in the generated one machine problems. We use END_NODE to indicate that the process on the node is interrupted and that the following node in the stack is processed, and END_UB to indicate that a desired duration solution was found. A new iteration of the search on the smallest eligible upper bound is then executed.

It is important to note that in the following example we do not use all elements of the method which would be too long, even for a small example. We will simply emphasize the key elements of the method. We consider the following seven task and two resource problem:

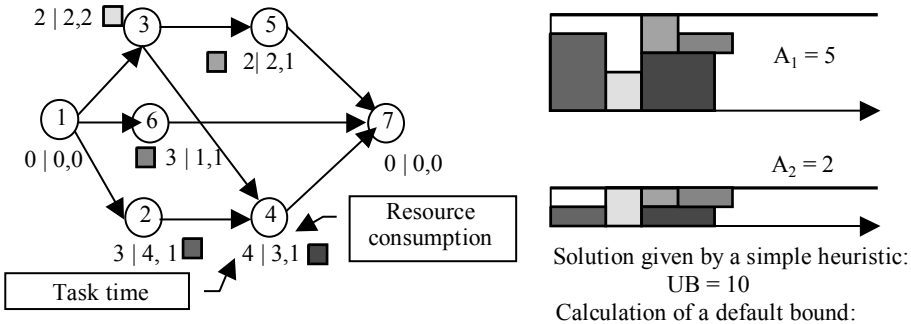


Figure 9.6. *Initialization of the tree search method*

The first iteration of the search algorithm for the smallest UB consists of verifying if there is a solution with a duration smaller than or equal to 9 = (10 + 8)/2. We briefly present the search tree route corresponding to this case. Nodes are scanned according to their number in the next diagram. The dotted arrows indicate backtracking in the search tree (Figure 9.8).

After this iteration, we have a solution lasting 9 with a lower bound of 8. We now need to verify if a solution lasting 8 exists.

As before, in Figure 9.8 the nodes are numbered in the order of their examination. The straight arrows indicate a descent in the search tree, the dotted arrows indicate backtracking. Since it was proven that no solution lasting 8 exists, at the end of this iteration we can conclude that the minimum duration for this example is 9. The solution previously presented for $UB_{test} = 9$ is therefore an optimal solution.

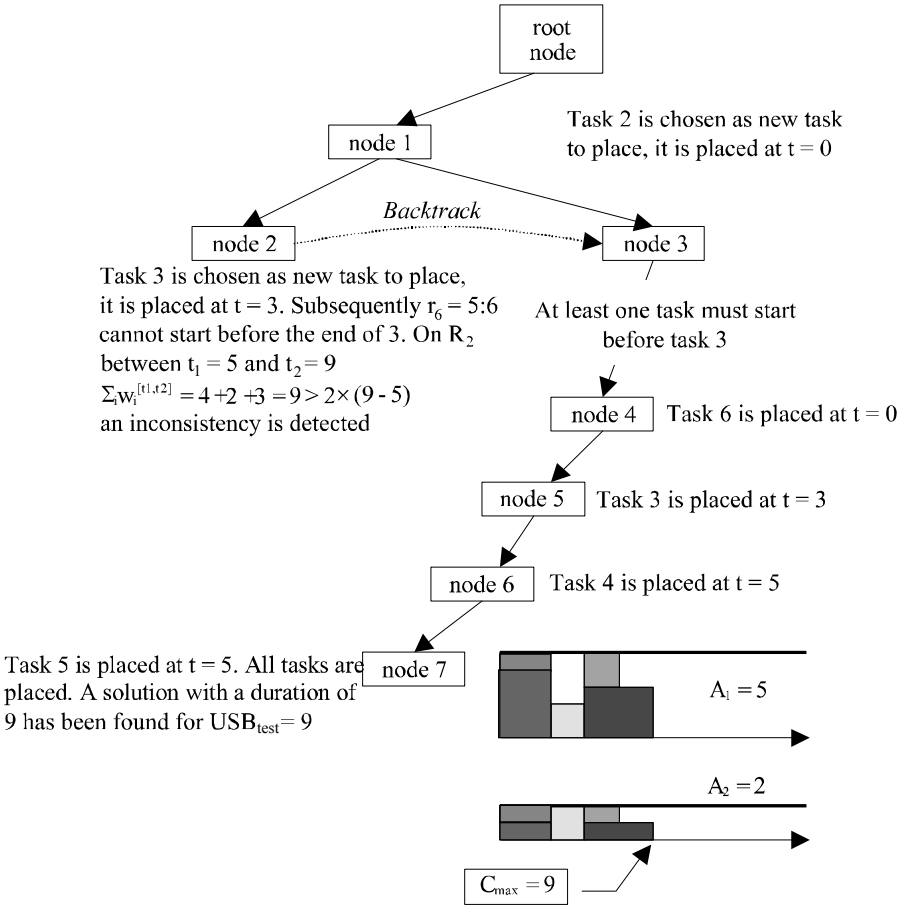


Figure 9.7. *Iteration for $UB_{test} = 9$*

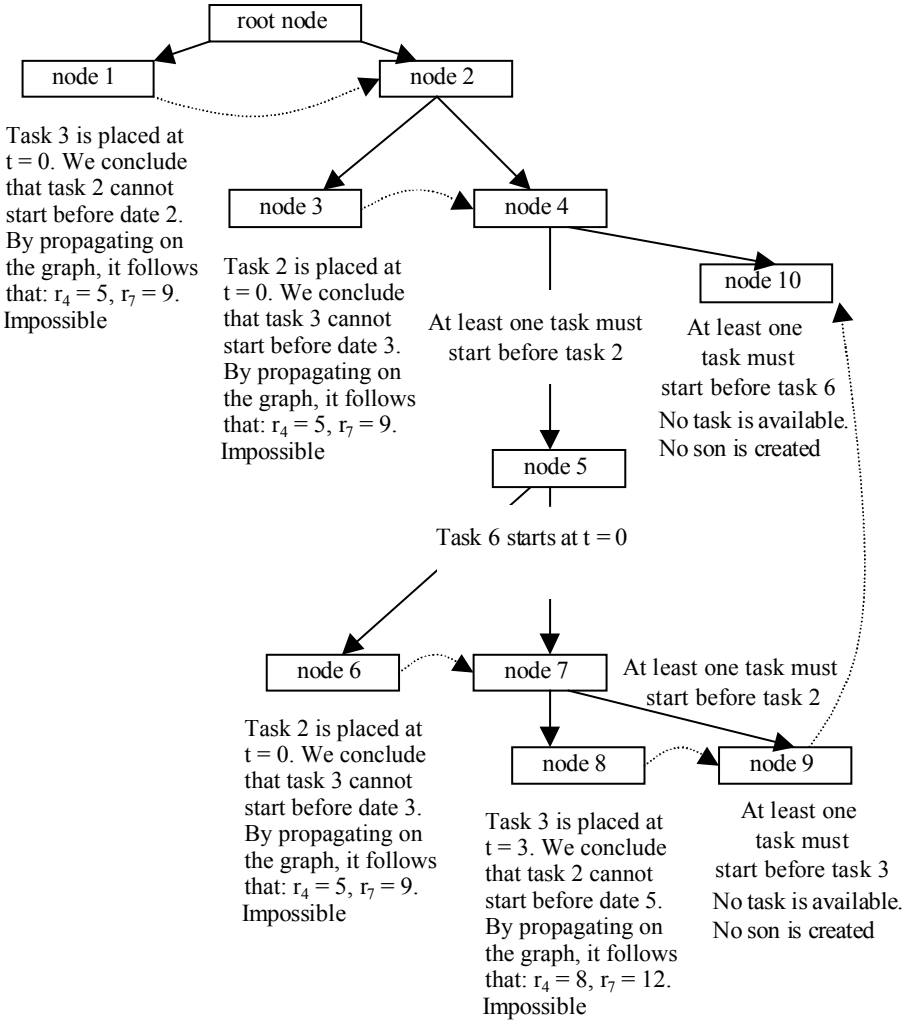


Figure 9.8. *Iteration for $UB_{test} = 8$*

9.3.3. *Results and fields of application of methods*

It is important to note that the results listed here are those supplied by the authors of referenced works. Great precautions must therefore be taken in comparing the different methods. In fact, most often, the test platform changes as well as the

language used. It is however interesting to attempt to highlight major trends in terms of efficiency and respective fields of application of these methods.

Patterson [PAT 84] proposed 110 RCPSP instances to compare the different exact methods. The size of these instances varies from 7 to 50 tasks (with an average of 22 tasks), with a number of resources from 1 to 3. Very good results were obtained with these instances, in particular with the DH method [DEM 92]. The authors compare this method to the one proposed by Stinson [STI 78]. A second version of the DH method [DEM 97] greatly improves these results: the 110 instances are resolved in an average time frame of 0.025 s. However, it is easy to produce the mainly disjunctive nature of the tests proposed by Patterson [BAP 97]. In addition, the DH method efficiency is strongly linked to an implementation developed with the cut-set rule [DEM 97].

An alternative was proposed by Kolish *et al.* [KOL 95], with the development of a configurable test generator, and the implementation of a library of tests mainly characterized by three factors accounting for the number of precedences, the average number of resources used for tasks, and the average quantity of resources used for each task according to the quantity of resources available. Once again, the DH method [DEM 97] is extremely efficient since it solves all 480 proposed tests. The method proposed by Mingozzi *et al.* [MIN 98] should also be mentioned as being powerful in these instances. However, it is interesting to note that these two methods use the same bound and have the cut-set rule in common, although they are not based on the same branching scheme.

Demeulemeester and Herroelen have emphasized the limits of the DH method. In fact, this method owes its efficiency in great part to the use of the cut-set rule which is very memory intensive. The authors even maintain that with this type of approach, 500 MB of memory is necessary to process tests with 62 tasks. The rule of dominance proposed by Sprecher [SPR 96], based on global left-shift, seems to be an alternative to the DH method to solve the tests proposed by Kolish *et al*. [KOL 95] without using the cut-set rule, and is thus less expensive.

The method proposed by Brucker *et al*. [BRU 98a] can also appear to be a good compromise between efficiency and memory space used. The results reported by the authors on the 480 tests from Kolish *et al*. are globally not as good as those mentioned previously, but generated tests performed on larger sizes (90 tasks) conclude that the method is highly efficient on large size instances. The bound used is based on the resolution of a linear program with the help of a column generation technique which requires consequent memory space. In addition, the method is clearly less powerful when resource consumption is low in relation to the availability of resources, which is the case with highly cumulative problems.

In this way, even though the parameters proposed by Kolish *et al*. for the characterization of the instances help us better understand their structure, these three factors do not completely account for their cumulative character. Baptiste and Le Pape [BAP 97] propose the comparison of different instances with the help of the disjunction ratio involving the number of task pairs which cannot be executed simultaneously, either because there is a relation of precedence in a broad sense between tasks, or because the sum of task consumption exceeds the resource availability. With this criteria, the authors have established a series of 40 reasonably sized highly cumulative instances (20 and 25 tasks): the ratio of disjunctions is on average equal to 0.33 for these instances, whereas it is 0.53 for the more cumulative instances proposed by Kolish *et al*., and 0.67 for instances proposed by Patterson. Baptiste and Le Pape [BAP 97] have shown that on traditional instances, using tools such as necessary existence conditions and adjustments based on time bound reasoning are not very efficient. On the contrary, these same tools ended up being essential in order to solve highly cumulative problems.

9.4. Conclusion

The goal of this chapter was to present, on the one hand, two multiple resource scheduling problems, whether these resources be disjunctive, made up of multiple copies (hybrid flow shop), or cumulative (RCPSP) and, on the other hand, traditional resolution methods for these two problems which are NP-hard.

For the hybrid flow shop, after a review of a few manufacturing applications related to this type of problem, we described the heuristic resolution methods based on simple principles: the determination of a list of jobs and allocation of these jobs on machines. We then explained an exact method (B&B), whose branching scheme is not explicitly based on the chronological development of schedules.

For the RCPSP, after a reminder of the structure of the problem, we have described in detail the different B&B components in other works. Finally, we have reported one of these methods where the branching scheme is based on the development of partial schedules in a simple example. Remember that the hybrid flow shop can be seen as a specific RCPSP case. The methods detailed for the RCPSP can be directly applied to the resolution of the hybrid flow shop and some of them are particularly efficient for this type of problem [NER 98].

In conclusion, we feel it is important to insist on the practical aspect linked to the resolution of these two types of problems. From the extremely abundant literature concerning these two problems, we can highlight simple heuristics to be implemented but where performance in terms of the quality of resulting solutions can be quite poor. Sophisticated heuristic methods (see Chapter 3) are most often a

way to reach a good compromise between the quality of resulting solutions and calculation time. Finally it is established that for reasonably sized problems, there are efficient exact methods for the resolution of the hybrid flow shop (up to 50 operations) as well as for the RCPSP (up to 60 tasks).

9.5. Bibliography

[ART 00] ARTIGUES C., ROUBELLAT F., "A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes", *European Journal of Operational Research*, vol. 127, no. 2, p. 297-316, 2000.

[ART 03] ARTIGUES C., MICHELON P., REUSSER S., "Insertion techniques for static and dynamic Resource-Constrained Project Scheduling", *European Journal of Operational Research*, vol. 149, no. 2, p. 249-267, 2003.

[ART 08] ARTIGUES Ch. DEMASSEY S. and NERON E*., Resource-Constrained Project Scheduling Models*, *Algorithms, Extensions and Applications*, ISTE, 2008.

[BAK 74] BAKER K.R., *Introduction to Sequencing and Scheduling*, Wiley, New York, 1974.

[BAP 97] BAPTISTE Ph. and LE PAPE C., "Constraint Propagation and Decomposition Techniques for Highly Disjunctive and Highly Cumulative Project Scheduling Problem", 3rd International Conference on Principles and Practice of Constraint Programming, in *Lecture Notes in Computer Science* 1330, Springer Verlag.

[BAP 99] BAPTISTE Ph., LE PAPE C. and NUIJTEN W., "Satisfiability Tests and Time Bound Adjustments for Cumulative Scheduling Problems", *Annals of Operations Research*, vol. 92, p. 305-333, 1999.

[BAP 04] BAPTISTE P., DEMASSEY S., "Tight LP bounds for Resource Constrained Project Scheduling", *OR Spectrum*, vol. 26, no. 2, p. 251-262, 2004.

[BOU 03] BOULEIMEN K., LECOCQ H., "A new efficient simulated annealing algorithm for the Resource-Constrained Project Scheduling Problem and its multiple mode version", *European Journal of Operational Research*, vol. 149, no. 2, p. 268-281, 2003.

[BRA 75] BRATLEY P., FLORIAN M. and ROBILLARD P., "Scheduling with Earliest Start and Due Date Constraints on Multiple Machine", *Naval Research Logistics Quarterly*, vol. 22, no. 1, p. 165-173, 1975.

[BRA 91] BRAH S.A. and HUNSUCKER J.L., "A Branch and Bound Algorithm for the Flowshop with Multiple processors", *European Journal of Operational Research*, vol. 51, p. 88-99, 1991.

[BRO 98] BROCKMANN K. and DANGELMAIER W., "A Parallel Branch and Bound Algorithm for Makespan Optimal Sequencing in Flow Shops with Parallel Machines", *Multiconference on Computational Engineering in Systems Applications (CESA'98), Symposium on Industrial and Manufacturing Systems, IEEE-SMC/IMACS*, p. 431-436, 1998.

[BRU 97] BRUCKER P., JURISCH B. and KRÄMER A., "Complexity of Scheduling Problems with Multi-purpose Machines", *Annals of Operations Research*, vol. 70, Scheduling: theory and applications, p. 57-73, 1997.

[BRU 98a] BRUCKER P., KNUST S., SCHOO A. and THIELE O., "A Branch and Bound Algorithm for the Resource-Constrained Project Scheduling Problem", *European Journal of Operational Research*, vol. 107, p. 272-288, 1998.

[BRU 98b] BRUCKER P. and KNUST S., A Linear Programming and Constraint Propagation-Based Lower Bound for the RCPSP, Internal Report, Osnabrück University, 1998.

[BRU 99] BRUCKER P., DREXL A., MOHRING R., NEUMANN K. and PESCH E., "Resource-Constrained Project Scheduling: Notation, Classification, Models and Methods", *European Journal of Operational Research*, vol. 112, p. 3-41, 1999.

[BRU 00] BRUCKER P., KNUST S., "A linear programming and constraint propagation-based lower bound for the RCPSP", *European Journal of Operational Research*, vol. 127, p. 355-362, 2000.

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

[BUT 73] BUTEN R.E. and SHEN V.Y., "A Scheduling Model for Computer Systems with Two Classes of Processors", *Saganiore Computer Conference on Parallel Processing*, p. 130-138, 1973.

[CAM 70] CAMPBELL H.G., DUDEK R.A. and SMITH M.L., "A Heuristic Algorithm for the n Job, m Machine Sequencing Problem", *Management Science*, vol. 16, no. 10, p. B630-B637, 1970.

[CAR 84] CARLIER J., "Problèmes d'Ordonnancement à Contraintes de Ressources: Algorithmes et Complexité", Doctoral thesis, Paris 6 University, 1984.

[CAR 88] CARLIER J. and CHRETIENNE P., *Problèmes d'Ordonnancement: Modélisation, Complexité, Algorithmes*, Masson, 1988.

[CAR 91] CARLIER J. and LATAPIE B., "Une Méthode Arborescente pour Résoudre les Problèmes Cumulatifs", *RAIRO – Recherche Opérationnelle*, vol. 25, no. 3, p. 311-340, 1991.

[CAR 97] CARLIER J. and NÉRON E., "An Exact Method for Solving the Multiprocessor Flowshop", *International Conference on Industrial Engineering and Production Management (IEPM'97)*, INRIA/INSA/University of Lyon 1, p. 592-600, 1997.

[CAR 98a] CARLIER J. and PINSON E., "Jackson Pseudo Preemptive Schedule for the $Pm/r_i,q_i/C_{max}$ Problem", *Annals of Operations Research*, vol. 83, p. 41-58, 1998.

[CAR 03] CARLIER J., NÉRON E., "On linear lower bounds for the Resource Constrained Project Scheduling Problem", *European Journal of Operational Research*, vol. 149, p. 314-324, 2003.

[CAR 07] CARLIER J., NÉRON E., "Computing redundant resources for the Resource Constrained Project Scheduling Problem", *European Journal of Operational Research*, vol. 176, p. 1452-1463, 2007.

[CHA 04] CHANG J., YAN W., SHAO H., "Scheduling a two-stage no-wait hybrid flowshop with separated setup and removal times", *Proceedings of the American Control Conference*, 2, p. 1412-1416, 2004.

[CHR 87] CHRISTOFIDES N., ALVAREZ-VALDES R. and TAMARIT J., "Project Scheduling with Resource Constraints: A Branch and Bound Approach", *European Journal of Operational Research*, vol. 29, p. 262-273, 1987.

[DAM 05] DAMAY J., *Techniques de résolution basées sur la programmation linéaire pour l'ordonnancement de projet*, PhD thesis, Blaise Pascal University, Clermont-Ferrand (France), 2005.

[DEA 91] DEAL D.E. and HUNSUCKER J.L., "The Two-Stage Flowshop Scheduling Problem with m Machines at Each Stage", *Journal of Information and Optimization sciences*, vol. 12, p. 407-471, 1991.

[DEB 06a] DEBELS D., DE REYCK B., LEUS R., VANHOUCKE M., "A hybrid scatter search/electromagnetismmeta-heuristic for project scheduling", *European Journal of Operational Research*, vol. 169, no. 2, p. 638-653, 2006.

[DEM 92] DEMEULEMEESTER E. and HERROELEN W., "A Branch and Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem", *Management Science*, vol. 38, no. 12, p. 1803-1818, 1992.

[DEM 94] DEMEULEMEESTER E., HERROELEN W., SIMPSON W.P., BAROUM S., PATTERSON J.H. and YANG K.K., "On a paper by Christofides *et al.* for Solving the Multiple-Resource Constrained, Single Project Scheduling Problem", *European Journal of Operational Research*, vol. 76, p. 218-228, 1994.

[DEM 96] DEMEULEMEESTER E. and HERROELEN W., "An Efficient Solution Procedure for the Preemptive Resource-Constrained Project Scheduling Problem", *European Journal of Operational Research*, vol. 90, p. 334-348, 1996.

[DEM 97] DEMEULEMEESTER E. and HERROELEN W., "New Benchmark Results for the Resource-Constrained Project Scheduling Problem", *Management Science*, vol. 43, p. 1485-1492, 1997.

[DEM 02b] DEMEULEMEESTER E., HERROELEN W., *Project Scheduling – A Research Handbook*, vol. 49 of International Series in Operations Research & Management Science, Kluwer Academic Publishers, 2002.

[DEM 05] DEMASSEY S., ARTIGUES C., MICHELON P., "Constraint propagation based cutting planes: an application to the resource-constrained project scheduling problem, *INFORMS Journal on Computing*, vol. 17, no. 1, p. 52–65, 2005.

[DER 98] DE REYCK B. and HERROELEN W., "A Branch-and-Bound Procedure for the Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations", *European Journal of Operational Research*, vol. 111, p. 152-174, 1998.

[DES 98] DESSOUKY M.M., DESSOUKY M.I. and VERMA S.K., "Flowshop Scheduling with Identical Jobs and Uniform Parallel Machines", *European Journal of Operational Research*, vol. 109, p. 620-631, 1998.

[DUP 98] DUPONT L., VIGNIER A., PORTMANN M-C. and PROUST C., "New Separation Schemes for Hybrid Flowshop", *Journal Européen des Systèmes Automatisés*, vol. 32, no. 4, p. 453-465, 1998.

[ELM 97] ELMAGHRABY S.E. and KARNOUB R.E., "Production Control in Hybrid Flowshops: an Example from Textile Manufacturing", Chapter 6 in *The Planning and Scheduling of Production Systems*, edited by A. Artiba and S.E. Elmaghraby, Chapman & Hall, 1997.

[ERS 91] ERSCHLER J., LOPEZ P. and THURIOT C., "Raisonnement Temporel sous Contraintes de Ressources et Problèmes d'Ordonnancement", *Revue d'Intelligence Artificielle*, vol. 5, no. 3, p. 7-32, 1991.

[ESQ 99] ESQUIROL P. and LOPEZ P., *L'Ordonnancement*, Economica, Paris, 1999.

[FLE 04] FLESZAR K., HINDI K., "Solving the Resource-Constrained Project Scheduling Problem by a variable neighborhood search", *European Journal of Operational Research*, vol. 155, p. 402-413, 2004.

[GOD 05] GODARD D., LABORIE P., NUIJTEN W., "Randomized large neighborhood search for cumulative scheduling", *Proceedings International Conference on Automated Planning and Scheduling* (ICAPS-05), Monterey, California, USA, 2005.

[GUI 96] GUINET A., SOLOMON M.M., KEDIA P.K. and DUSSAUCHOY A., "A Computational Study of Heuristics for Two-Stage Flexible Flowshops", *International Journal of Production Research*, vol. 34, vol. 5, p. 1399-1415, 1996.

[GUP 88] GUPTA J.N.D., "Two-Stage Hybrid Flowshop Scheduling Problems", *Journal of Operational Research Society*, vol. 39, no. 4, p. 359-364, 1988.

[GUP 91] GUPTA J.N.D. and TUNC E.A., "Schedules for a Two-Stage Hybrid Flowshop with Parallel Machines at the Second Stage", *International Journal of Production Research*, vol. 29, no. 7, p. 1489-1502, 1991.

[GUP 94] GUPTA J.N.D. and TUNC E.A., "Scheduling a Two-Stage Hybrid Flowshop with Separable Setup and Removal Times", *European Journal of Operational Research*, vol. 7, p. 415-428, 1994.

[GUP 95] GUPTA J.N.D., HARIRI A.M.A. and POTTS C.N., "Scheduling a Two-Stage Hybrid Flowshop with Parallel Machines at the First Stage", *Annals of Operations Research*, vol. 69, Mathematics of Industrial Scheduling II, p. 171-191, 1997.

[GUP 98] GUPTA J.N.D., TUNC E.A., "Minimizing tardy jobs in a two-stage hybrid flowshop", *International Journal of Production Research*, 36(9), p. 2397-2417, 1998.

[HAO 97] HAOUARI M. and M'HALLAH R., "Heuristic Algorithms for the Two-Stage Hybrid Flowshop Problem", *Operations Research Letters*, vol. 21, p. 43-53, 1997.

[HE 07] HE L., SUN S., LUO R., "A hybrid two-stage flowshop scheduling problem", *Asia-Pacific Journal of Operational Research*, 24(1), p. 45-56, 2007.

[HEN 98a] HERROELEN W., DEMEULEMEESTER E. and DE REYCK B., "Resource-Constrained Project Scheduling – A Survey of Recent Developments", *Computer and Operations Research*, vol. 25, no. 4, p. 279-302, 1998.

[HEN 98b] HERROELEN W., DEMEULEMEESTER E. and DE REYCK B., "A Classification Scheme for Project Scheduling", *Sixth International Workshop on Project Management and Scheduling*, WPMS-98, p. 67-70, 1998.

[JIN 06] JIN Z., YANG Z., ITO T., "Metaheuristic algorithms for the multistage hybrid flowshop scheduling problem", *International Journal of Production Economics*, 100(2), p. 322-334, 2006.

[JOH 54] JOHNSON S.M., "Optimal Two and Three-Stage Production Schedules with Set-up Times Included", *Naval Research Logistics Quarterly*, vol. 1, no. 1, p. 61-68, 1954.

[JUR 92] JURISCH B., Scheduling Jobs in Shops with Multi-Purpose Machines, PhD Thesis, Osnabrück University, Germany, 1992.

[KLE 99] KLEIN R. AND SCHOLL A., "Computing lower bounds by destructive improvement: An application to resource-constrained project scheduling", *European Journal of Operational Research*, vol. 112, p. 332-346, 1999.

[KLE 00a] KLEIN R., "Bidirectional planning: improving priority rule-based heuristics for scheduling resource-constrained projects", *European Journal of Operational Research*, vol. 127, p. 619-638, 2000.

[KOC 03] KOCHETOV Y., STOLYAR A., "Evolutionary local search with variable neighborhood for the Resource Constrained Project Scheduling Problem", *Proceedings of the 3rd International Workshop of Computer Science and Information Technologies* (CSIT'03), 2003.

[KOL 95] KOLISH R., SPRECHER A. and DREXL A., "Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problem", *Management Science*, vol. 41, no. 10, p. 1693-1703, 1995.

[KOL 97] KOLISH R. and PADMAN R., "An Integrated Survey of Project Scheduling", Internal Report, University of Kiel, 1997.

[KOL 06] KOLISCH R., HARTMANN S., "Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update", *European Journal of Operational Research*, vol. 174, no. 1, p. 23–37, 2006.

[LAB 05] LABORIE P., "Complete MCS-Based Search: Application to Resource Constrained Project Scheduling", Proceedings *of the International Joint Conference on Artificial Intelligence*, (IJCAI'05), Edinburgh, UK, 2005.

[LAH 82] LAHRICHI A., "Ordonnancements. La Notion de 'Parties Obligatoires' et son Application aux Problèmes Cumulatifs", *RAIRO – Recherche Opérationnelle*, vol. 16, p. 241-262, 1982.

[LAH 07] LAHA, D., CHAKRABORTY, U.K., "An efficient stochastic hybrid heuristic for flowshop scheduling", *Engineering Applications of Artificial Intelligence*, 20(6), 851-856, 2007.

[LEE 93] LEE C., CHENG T.C.E. and LIN B.M.T., "Minimizing the Makespan in the 3-Machine Assembly Flowshop Scheduling Problem", *Management Science*, 35(3), p. 616-625, 1993.

[LEE 94] LEE C.Y. and VAIRAKTARAKIS G., "Minimizing Makespan in Hybrid Flowshops", *Operations Research Letters*, 16(3), p. 149-158, 1994.

[LEE 04] LEE G.C. and KIM Y-D., "A branch-and-bound algorithm for a two-stage hybrid flowshop scheduling problem minimizing total tardiness", *International Journal of Production Research*, 42(22), p. 4731-4743, 2004.

[LI 97] LI S., "A Hybrid Two-Stage Flowshop with Part Family, Batch Production, Major and Minor Setups", *European Journal of Operational Research*, 102, p. 142-156, 1997.

[LOP 92] LOPEZ P., ERSCHLER J. and ESQUIROL P., "Ordonnancement de Tâches sous Contraintes: une Approche Energétique", *RAIRO Automatique, Productique, Informatique Industrielle*, vol. 26, no. 6, p. 453-481, 1992.

[LOW 08] LOW C., HSU C-J. and SU C-T., "A two-stage hybrid flowshop scheduling problem with a function constraint and unrelated alternative machines", *Computers and Operations Research*, 35(3), p. 845-853, 2008.

[MER 02] MERKLE D., MIDDENDORF M. and SCHMECK H., "Ant colony optimization for Resource-Constrained Project Scheduling", *IEEE Transactions on Evolutionary Computation*, vol. 6, p. 333-346, 2002.

[MIN 98] MINGOZZI A., MANIEZZO V., RICCIARDELLI S. and BIANCO L., "An Exact Algorithm for Project Scheduling with Resource Constraints based on New Mathematical Formulation", *Management Science*, vol. 44, p. 714-729, 1998.

[MÖH 84] MÖHRING R. H., "Minimizing Costs of Resource Requirements in Project Networks Subject to a Fixed Completion Time", *Operations Research*, vol. 32, p. 89-120, 1984.

[MÖH 98] MOHRING R.H., STORK F. and UETZ M., "Resource-Constrained Project Scheduling with Time Windows: A Branching Scheme Based on Dynamic Release Dates", Internal Report, Berlin Technical University, 1998.

[MOR 05] MORITA H. and SHIO N., "Hybrid branch and bound method with genetic algorithm for flexible flowshop scheduling problem", *JSME International Journal, Series C: Mechanical Systems, Machine Elements and Manufacturing*, 48(1), p. 46-52, 2005.

[MOU 99] MOURSLI O., Scheduling the Hybrid Flowshop: Branch and Bound Algorithm, PhD Thesis, Louvain Catholic University, Belgium, 1999.

[MOU 00] MOURSLI O., POCHET Y., "Branch-and-bound algorithm for the hybrid flowshop", *International Journal of Production Economics*, 64(1), p. 113-125, 2000.

[NAW 83] NAWAZ M., ENSCORE E. and HAM I., "A Heuristic Algorithm for the m machine n Job Flowshop Sequencing Problem", *OMEGA*, vol. 11, no. 1, p. 91-95, 1983.

[NER 98] NÉRON E., BAPTISTE PH., CARLIER J. and LE PAPE C., "Global Operations for the Multi-Processor Flow-Shop", *6th International Workshop on Project Management and Scheduling (PMS'98)*, p. 248-251, 1998.

[NER 99] NÉRON E., Du Flow-Shop Hybride au Problème Cumulatif, PhD Thesis, Compiègne University of Technology, 1999.

[NOW 98] NOWICKI E. and SMUTNICKI C., "The Flow Shop With Parallel Machines: a Tabu Search Approach", *European Journal of Operational Research*, vol. 106, p. 226-253, 1998.

[OLI 99] OLIVER A., VERRIERE J-C., PROUST C. and BILLAUT J.-C., Résolution de Problèmes d'Ordonnancement de Type Flow-Shop Hybride par des Procédures par Séparation et Evaluation, Technical report no. 7875, E3i, University of Tours, 1999.

[OZD 95] OZDAMAR L. and ULUSOY G., "A Survey on the Resource Constrained Project Scheduling Problem", *IIE Transactions*, vol. 27, p. 574-586, 1995.

[PAL 04] PALPANT M., ARTIGUES C., MICHELON P., "LSSPER: Solving the Resource-Constrained Project Scheduling Problem with Large Neighbourhood Search", *Annals of Operations Research*, vol. 131, no. 1-4, p. 237–257, 2004.

[PAT 74] PATTERSON J.H and HUBER W.D, "A Horizon-Varying Zero-One Approach to Project Scheduling", *Management Science*, vol. 20, no. 6, p. 990-998, 1974.

[PAT 76] PATTERSON J.H and ROTH G., "Scheduling a Project Under Multiple Resource Constraints: a Zero-one Programming Approach", *AIIE Transactions*, vol. 8, no. 4, p. 449-456, 1976.

[PAT 84] PATTERSON J.H., "A Comparison of Exact Approaches for Solving the Multiple Constrained Resource Project Scheduling Problem", *Management Science*, vol. 30, no. 7, p. 854-867, 1984.

[PAT 90] PATTERSON J.H., SLOWINSKI R., TALBOT F.B. and WEGLARZ J., "Computational Experience with a Backtracking Algorithm for Solving a General Class of Precedence and Resource-Constrained Scheduling Problems", *European Journal of Operational Research*, vol. 49, p. 68-79, 1990.

[PIN 94] PINSON E., PRINS C., RULLIER F., "Using tabu search for solving the Resource-Constrained Project Scheduling Problem", *Proceedings of the Fourth International Workshop on Project Management and Scheduling (PMS'94)*, Leuven, Belgium, 1994.

[POR 98] PORTMANN M-C., VIGNIER A., DARDILHAC D. and DEZALAY D., "Some hybrid flow-shops scheduling by crossing branch and bound and genetic algorithms", *European Journal of Operational Research*, vol. 107, p. 389-400, 1998.

[RIA 98] RIANE F., ARTIBA A. and ELMAGHRABY S.E., "A hybrid three-stage flowshop problem: efficient heuristics to minimize makespan", *European Journal of Operational Research*, vol. 109, p. 321-329, 1998.

[RIC 98] RICHARD P., BARRIER J. and PROUST C., "Economical aspects of short-term planning in the bottle-glass industry", *Revue VERRE*, vol. 4, no. 4, p. 7-14, 1998.

[RUI 08] RUIZ R., SERIFOGLU F.S., URLINGS T., "Modeling realistic hybrid flexible flowshop scheduling problems", *Computers and Operations Research*, 35 (4), p. 1151-1175, 2008.

[SAN 95] SANTOS D.L., HUNSUCKER J.L. and DEAL D.E., "Global Lower Bounds For Flowshops with Multiple Processors", *European Journal of Operational Research*., vol. 80, p. 112-120, 1995.

[SHE 72] SHEN V.Y. and CHEN Y.E., "A scheduling strategy for the flowshop problem in a system with two classes of processors", *Conference on Information and Systems Science*, p. 645-649, 1972.

[SPR 94] SPRECHER A., "Resource-Constrained Project Scheduling: Exacts Methods for the Multi-Mode Case", *Lectures Notes in Economics and Mathematical Systems*, no. 409, Springer Verlag, 1994.

[SPR 95] SPRECHER A., KOLISH R. and DREXL A., "Semi-Active, Active and Non-Delay Schedules for the Resource-Constrained Project Scheduling Problem", *European Journal of Operational Research*, vol. 80, no. 1, p. 94-102, 1995.

[SPR 96] SPRECHER A., Solving the RCPSP Efficiently at Modest Memory Requirements, Internal Report no. 425, Kiel University, 1996.

[SRI 89] SRISKANDARAJAH C. and SETHI S.P., "Scheduling algorithms for flexible flowshops: worst and average case performance", *European Journal of Operational Research*, vol. 43, p. 143-160, 1989.

[STI 78] STINSON J.P., DAVIS E.W. and KHUMAWALA B.M., "Multiple Resource-Constrained Scheduling using Branch and Bound", *AIIE Transactions*, vol. 10, p. 252-259, 1978.

[TKI 00] T'KINDT V., BILLAUT J-C. and HOUNGBOSSA H., "A multi-criteria heuristic to solve a 2-stage hybrid flowshop scheduling problem", *Journal Européen des Systèmes Automatisés*, 34(9), p. 1187-1200, 2000.

[TOW 77] TOWNSEND W., "Sequencing n jobs on m machines to minimize maximum tardiness: a branch and bound solution", *Management Science*, vol. 23, no. 9, p. 1016-1019, 1977.

[VAL 03] VALLS V., QUINTANILLA M., BALLESTIN F., "Resource-Constrained Project Scheduling: A Critical Reordering Heuristic", *European Journal of Operational Research*, vol. 149, p. 282-301, 2003.

[VAL 04] VALLS V., QUINTANILLA M., BALLESTIN F., "A population-based approach to the Resource-Constrained Project Scheduling Problem", *Annals of Operations Research*, vol. 131, p. 305-324, 2004.

[VAN 94] VANDEVELDE A., Minimizing the makespan in a multiprocessor flowshop, Master's thesis, Eindhoven, 1994.

[VIG 97] VIGNIER A., Contribution à la résolution des problèmes d'ordonnancement de type monogamme multimachine (flowshop hybride), Doctoral Thesis, University of Tours, 1997.

[VIG 99] VIGNIER A., BILLAUT J-C. and PROUST C., "Les problèmes de flowshop hybride: état de l'art", *RAIRO RO*, vol. 33, no. 2, p. 117-183, 1999.

[WEG 79] WEGLARZ J., "Project Scheduling with Discrete and Continuous Resources", *IEEE Transactions on SMC*, vol. 9, p. 644-650, 1979.

[WEG 05] WEGLARZ J. and JOZEFOWSKA J, *Perspectives in Modern Project Scheduling*, International Series in Operations Research & Management Science, vol. 92., Springer, 2005.

[WON 01] WONG W.K., CHAN C.K., IP W.H., "A hybrid flowshop scheduling model for apparel manufacture", *International Journal of Clothing Science and Technology*, 13(2), p. 115-131, 2001.

This page intentionally left blank

Chapter 10

Open Shop Scheduling

10.1. General overview

In production scheduling problems called *open shop* problems (or problems *with free job routes*), each item to be produced must have several operations on machines but in a totally open sequence. The absence of imposed order between operations makes these problems extremely combinatorial and prevents the application of already proven flow shop and job shop resolution techniques. However, it is necessary to have powerful resolution methods for these problems that have long been neglected, because they are encountered much more often because of the fact that flexible shops are increasingly common.

The main problem is a shop problem called open shop. This is why the bulk of this chapter is focused on this problem. Section 10.2 presents open shop in the context of shop problems and gives a few applications. The complexity of the different versions of this problem is discussed in section 10.3. In particular, contrary to flow shop and job shop, open shop becomes easy when operations can be interrupted: this remarkable case, actually encountered in telecommunications, deserves to be explained in section 10.4. "Simple" heuristics (excluding metaheuristics) are described in section 10.5. The other resolution methods using the disjunctive model are reviewed in section 10.6, and metaheuristics, easier to present, are presented in section 10.7. In section 10.8 we present exact methods. The available tests and comparative algorithm performances are described in section 10.9. The last section (10.10) presents a few neighboring problems encountered outside of production, mainly in telecommunications.

Chapter written by Christian PRINS.

10.2. The open shop problem

10.2.1. *Open shop in relation to other shop problems*

In shop problems, a set T of n jobs T_1, T_2, ..., T_n must be executed over a set M of m specialized machines M_1, M_2, ..., M_m. Simply put, we consider that if each job T_i (in practice a product to be manufactured) contains m operations (i, j), $j = 1, 2, ... m$ the operation (i, j) occurs on machine M_j. Even with this hypothesis, shop problems become difficult with three machines. However, most resolution methods in the chapter are adaptable to jobs with more than m operations. Matrix P, $n \times m$, gives the integer processing time p_{ij} of each operation (i, j). There are two types of common constraints: a machine cannot process more than one job at a time, and a job cannot occur on more than one machine at the same time.

A schedule is defined by allocating a start date t_{ij} (or an end date $C_{ij} = t_{ij} + p_{ij}$) to each operation (i, j) while respecting constraints. The most common objective is to minimize the total duration C_{max} (maximum of completion times), also called makespan. In the following, C^*_{max} denotes the minimal makespan, p_i the duration of job T_i if executed without pause (sum of elements in row i of P), and Z_j the workload of machine M_j (sum of j column elements). According to constraints, these quantities are lower bounds of the makespan for any schedule. In particular, their maximum, LB, is a traditional bound, valid for any shop problem:

$$LB = \max\left(\max_{j=1,m}\left(\sum_{i=1,n} p_{ij}\right), \max_{i=1,n}\left(\sum_{j=1,m} p_{ij}\right)\right) \qquad [10.1]$$

If we voluntarily delay operations, we have an unlimited number of possible schedules. A schedule is *semi-active* if no operation can be launched earlier while conserving its rank in its job or on its machine. A semi-active schedule is *active* if no operation can be started earlier without violating the constraints or delaying another operation (in this way, no operation can be placed in a possible "gap" located before it on its machine). Finally, an active schedule is called *non-delay* or *dense* if no machine is left idle when it could process an operation.

Semi-active schedules include active schedules, which in turn contain non-delay schedules and these three types of schedules are *finite in number*. A traditional result stipulates that there is always an optimal active schedule in the case of makespan minimization. A resolution algorithm can then limit its search to only explore active solutions in order to find this *optimum*. On the other hand, the set of non-delay schedules (the smallest one) may not contain an optimal solution.

The two most well-known shop problems are flow shop and job shop. They have been studied since the 1950s and are characterized by a fixed route for each job (see Chapter 2). In the *general flow shop* modeling linear structure shops, products (jobs) follow the same route M_1, M_2, ..., M_m but a job may overtake another. The *permutation flow shop* models the specific case of assembly lines: products cannot overtake each other, and a sequence of jobs is enough to define a schedule. In job shop, each job has its own route to follow: the flow shop is therefore a specific job shop case, with identical routes. Job shop problems can be found in shops organized in islands, with diversified products (small mechanics, for example).

The open shop is the case where the order of operations for each job is completely open. An instance is thus defined by n, m and P. The problem appeared late (around 1975) in scheduling literature, because it is less frequent than flow shop and job shop. However, many sequencing constraints are artificial in reality: they are not induced by the product, but often by a rigid line organization in the shop. In modern flexible shops, we have many more choices in the order of operations. Early in the 1980s, the satellite telecommunications field raised completely unexpected open shop type problems. All this makes open shop problems increasingly common in the industry.

10.2.2. *An example*

Consider a small example with three jobs and three machines to illustrate the differences between open shop and other shop problems. Table 10.1 gives the P matrix of operation times with machine-loads and job times. Any feasible schedule will last at least 1,000, the LB value. Figure 10.1 is the graphical representation (Gantt chart) of a permutation flow shop schedule defined by the sequence of jobs (1, 2, 3): the makespan is 1,358. We observe inactivity periods on M_2 and M_3, typical of this problem. Downstream machines wait for the products to reach them. Figure 10.2 shows an open shop schedule. The free routes of jobs facilitate the problem, hence total time is decreased to 1,239. This schedule can still be improved by moving job T_1 to date 85 on M_3. The resulting schedule is optimal since the LB bound is reached.

	M_1	M_2	M_3	p_i
T_1	97	72	500	669
T_2	261	540	85	886
T_3	642	274	84	1,000
Z_j	1,000	886	669	***1,000***

Table 10.1. *3 × 3 example with bound calculation*

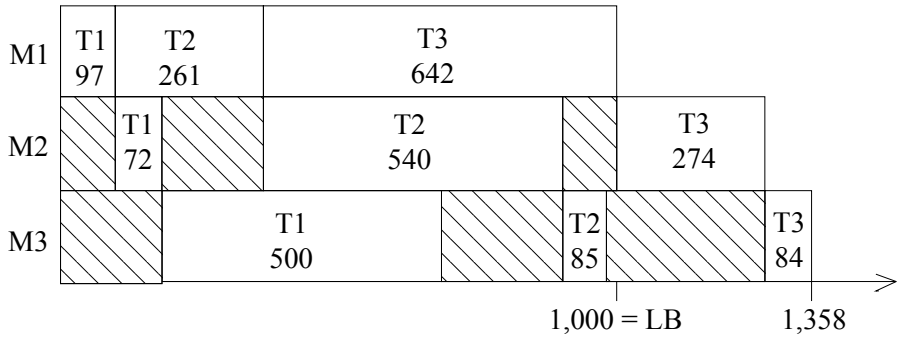


Figure 10.1. *Permutation flow shop type scheduling example*

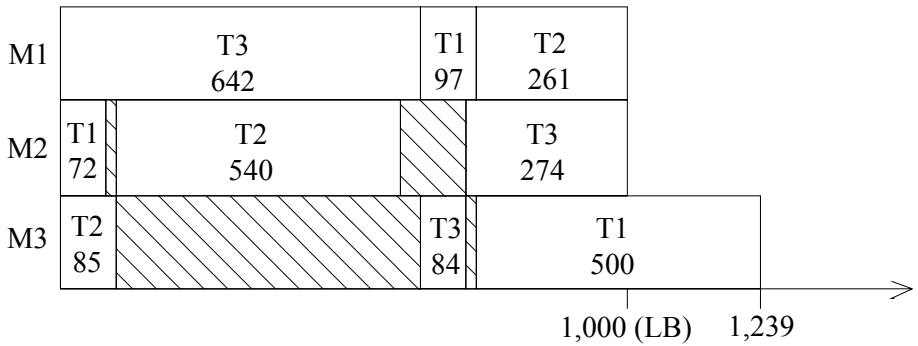


Figure 10.2. *Open shop type scheduling example*

10.2.3. *A few real open shop examples*

Table 10.2 presents four examples of open shop applications. Outside of production, the problem is encountered in testing and maintenance activities, in service and telecommunications sectors. The telecommunications context is explained in section 10.10. Note that in machining, for example, the Gantt chart line scan corresponds to machines or parts. In fact, in an open shop environment, jobs and machines are often found by convention and play an equal role. In this way, if we transpose matrix *P*, the optimal time does not change, contrary to flow shop and job shop.

Application	Jobs	Operations	Machines
Machining	Part machining	Operation to execute on a part	Machine-tools
Equipment test	Equipment to test	Basic test	Operator or Test bed
Examination of patients in hospital	Patient with routine examination to perform	Patient tests: analyses, x-rays, etc.	Doctors or specialized rooms
Digital satellite telecommunications	Group of packets to transmit by one Earth-station	Packet from a transmitting station to a repeater	Satellite repeaters (channels)

Table 10.2. *Some real open shop examples*

10.3. Complexity of open shop problems

10.3.1. *Overview*

With no specific hypothesis, open shop is an NP-hard problem from three machines. However, as with flow shop and job shop, it becomes polynomial for $m = 2$. If operation preemptions (interruptions) are authorized, the open shop also becomes polynomial, but, other shop problems remain NP-hard. For these two polynomial cases, $C^*_{max} = LB$. All these results were established by Gonzalez and Sahni [GON 76]. Since then, a multitude of papers on the complexity of particular cases have emerged, but few studies have been completed on practical resolution methods, which is typical for a relatively recent problem. The next sections present a few particular cases.

10.3.2. *Polynomial geometric methods*

Let p_{max} denote the longest operation time and Z_{max} the biggest machine-load. The open shop problem becomes polynomial with $C^*_{max} = LB$ if the p_{max} / Z_{max} ratio is small. These results come from studies in geometry on a problem called *compact summation of vectors*. Given a series of m vectors in $\mathbb{R}^n$, in what sequence should their sum be executed in order for the partial sums to be confined in the smallest sphere possible? This problem is NP-hard but different bounds are known for the sphere's radius and good use has been made of them in scheduling.

Results of this type are summarized in a review [SEV 94]. Here are two examples. Fiala in 1982 calls m' the smallest power of $2 \geq m$ and shows that, if $Z_{max} \geq (16.m'.\log_2 m' + 5.m').p_{max}$, then the open shop can be solved in $O(n^2m^3)$.

However, operations must be very small compared to machine-loads: for $m = 3$, we find $Z_{max}/p_{max} \geq 148$. Sevast'janov refined these bounds in 1990: the problem can be solved in $O(n^2m^2)$ if $Z_{max} \geq (m^2 - 1 + 1/(m-1)).p_{max}$. For $m = 3$, this corresponds to $Z_{max}/p_{max} \geq 8.5$. However, the probability of finding open shops verifying these conditions in practice becomes very low when m increases.

10.3.3. *The polynomial m = 2 case*

The report from Gonzalez and Sahni [GON 76] also gives a polynomial algorithm reaching *LB* for the two machine case (or two jobs since the optimal makespan of an open shop does not change when *P* is transposed). In 1995, Pinedo [PIN 95] proposed a simpler algorithm based on a priority rule called LAPT (longest alternate processing time). We begin at time 0 with idle machines. At each iteration, we search for the smallest time t to which a machine can start an operation. We then execute at t the job with the longest operation remaining on the other machine. If the schedule has no down time, it is clearly optimal because its makespan is equal to the maximum *LB* of the two machine-loads. Otherwise, Pinedo shows that the algorithm will, at most, leave a gap on only one machine and that this gap does not delay the schedule.

	M_1	M_2	p_i
T_1	97	72	169
T_2	261	540	801
T_3	642	274	916
Z_j	1,000	886	***1,000***

Table 10.3. *A two machine example*

Table 10.3 illustrates an example where $n = 3$, $m = 2$ and $LB = 1{,}000$. Figure 10.3 shows the Gantt chart of the schedule obtained by the LAPT rule: *LB* bound is actually reached. We detail the algorithm sequence. At the beginning, M_1 and M_2 are idle at $t = 0$. (3, 2) is the executable operation at $t = 0$ whose job has the longest operation on the other machine. We place (3, 2) on M_2. Then, M_1 can start two operations at $t = 0$: (1, 1) and (2, 1). Job T_2 is the one with the largest remaining work (540) on the other machine, M_2. We then place (2, 1) on M_1. Subsequently, M_1 can only start one operation at $t = 261$, (1, 1). We therefore have no choice and we place this operation. We can then only execute (2, 2) when M_2 is ready at $t = 274$, then (3, 1), and (1, 2).

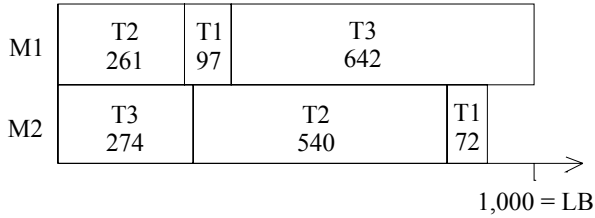


Figure 10.3. *Scheduling found by optimal LAPT rule*

10.3.4. *The boundary m = 3 case*

We say that a machine M_j is *dominated* by another M_k if the longest operation of M_j does not last longer than the shortest operation of M_k. In other words, if max (p_{ij}, $i = 1 \ldots n$) ≤ min (p_{ik}, $i = 1 \ldots n$). If one of the three machines is dominated, the open shop can be solved in $O(n)$ and C^*_{max} can exceed *LB* [ADI 89]. Roughly, the algorithm first calculates a schedule for the two non-dominated machines, then it places dominated machine operations. The case where jobs only have two operations, where one has to be on M_1 (M_1 is then called *bottleneck machine*) can also be handled in $O(n)$, with a makespan which can exceed *LB* [DRO 99]. The bottleneck machine case has an open complexity.

10.3.5. *Special open shops*

In the case of makespan minimization, the smallest additional constraint deletes the open shop polynomial cases. In this way, the $m = 2$ case becomes NP-hard when we add release dates to jobs [LAW 81] or if jobs must be accomplished without wait time between operations (no-wait case), as in the steel industry [SAH 79]. In the preemptive case, it becomes NP-hard if one additional resource is added, whether it is renewable [WER 91] or consumable [WER 92]. For other optimization criteria, the open shop is generally NP-hard from $m = 2$, for example, to minimize the average job completion time or mean flow time [ACH 82] or the maximum lateness in case of job due dates [LAW 81].

10.4. The preemptive case (operations executable multiple times)

10.4.1. *Gonzalez and Sahni algorithm*

It is remarkable that open shop is “easy” in the preemptive case, as opposed to flow shop and job shop. The corresponding algorithm [GON 76] deserves a detailed explanation, especially since the problem is currently present in

telecommunications. Its complexity is $O(m^2n^2)$, actually $O(r^2)$ if there are $r \leq mn$ non-zero operations. To simplify the presentation, we use a "square" open shop ($m = n$). We first give the algorithm text before explaining it with an example.

Add fictitious activity to P to make all sums
for rows and columns equal to LB (matrix P')
while P' is not zero **do**
    Search for a perfect matching C in the bipartite graph
    associated with P' (n operations, one per row and per column)
    $D \leftarrow \min\{p'_{ij} \mid (i,j) \in C\}$
    Build a partial schedule (schedule slice) where each
    operation (i, j) of C is executed for D units of time.
    for any operation $(i, j) \in C$ **do**
        $p'_{ij} \leftarrow p'_{ij} - D$
Remove fictitious activity from scheduling obtained.

Algorithm 10.1. *Gonzalez and Sahni algorithm for the preemptive case*

10.4.2. *An example*

In Table 10.4, we use our 3 × 3 example. We add 114 to (1, 2), 217 to (1, 3) and 114 to (2, 3) to obtain matrix *P'* where all line and column sums equal 1,000 (Table 10.5). We speak of a "quasi-bistochastic" matrix.

	$\mathbf{M_1}$	M_2	M_3	p_i
T_1	97	72	500	669
T_2	261	540	85	886
T_3	642	274	84	1,000
Z_j	1,000	886	669	***1,000***

	M_1	M_2	M_3	p_i
T_1	97	186	717	1,000
T_2	261	540	199	1,000
T_3	642	274	84	1,000
Z_j	1,000	1,000	1,000	***1,000***

Tables 10.4 and 10.5. *3 × 3 example and its quasi-bistochastic matrix P'*

We extract from *P'* a set of operations which can be executed in parallel, called *matching*. In *P'* it corresponds to non-zero elements placed one per row and per column. A matching is said to be *perfect* if it contains n operations. Gonzalez and Sahni proved that a perfect matching always exists in *P'*, even with the presence of zero elements. An algorithm for finding such matching can be found in [PRI 94a]. We can, for example, use {(1, 3), (2, 2), (3, 1)}, with $D = \min\{717, 540, 642) =$

540. We execute these operations (bordered in Table 10.5) for 540 units of time to obtain a partial schedule (or schedule slice) and we update *P'* (Table 10.6).

	M_1	M_2	M_3	p_i
T_1	97	186	177	460
T_2	261	0	199	460
T_3	102	274	84	460
Z_j	460	460	460	***460***

	M_1	M_2	M_3	p_i
T_1	97	186	0	283
T_2	84	0	199	283
T_3	102	97	84	283
Z_j	283	283	283	***283***

Tables 10.6 and 10.7. *P' after first and second matching extraction*

P' remains quasi-bistochastic, we have taken out 540 from each row and each column. On the other hand, the remaining matrix bound is now $LB - D = 1{,}000 - 540 = 460$. Due to the existence of a perfect matching at each iteration, the original matrix *P'* will be able to be broken down into a sequence of schedule slices whose total duration will be *LB*. This process converges because at least one operation cancels itself in *P'* at each matching extraction, the operation with processing time *D*.

Continuing with matching {(1, 3), (2, 1), (3, 2)}, and $D = \min\{177, 261, 274\}$, or 177. We execute these operations for 177 units of time and we update *P'* (Table 10.7). Then, for example, we remove {(1, 2), (2, 3), (3, 1)} with $D = 102$ (Table 10.8), then {(1, 1), (2, 3), (3, 2)} with $D = 97$ (Table 10.9), then {(1, 2), (2, 1), (3, 3)} with $D = 84$. The algorithm is finished because *P'* is zero.

	M_1	M_2	M_3	p_i
T_1	97	84	0	181
T_2	84	0	97	181
T_3	0	97	84	181
Z_j	181	181	181	***181***

	M_1	M_2	M_3	p_i
T_1	0	84	0	84
T_2	84	0	0	84
T_3	0	0	84	84
Z_j	84	84	84	***84***

Tables 10.8 and 10.9. *P' after third and fourth matching extraction*

Figure 10.4 illustrates the breakdown which actually lasts *LB* and contains five matchings here. It also gives the valid schedule for *P*, obtained by removing the fictitious activity. Note that most operations are not fragmented, which raises the minimization problem of the number of preemptions while keeping a makespan

equal to *LB*. This problem is NP-hard, but we can decrease preemptions in a heuristic manner.

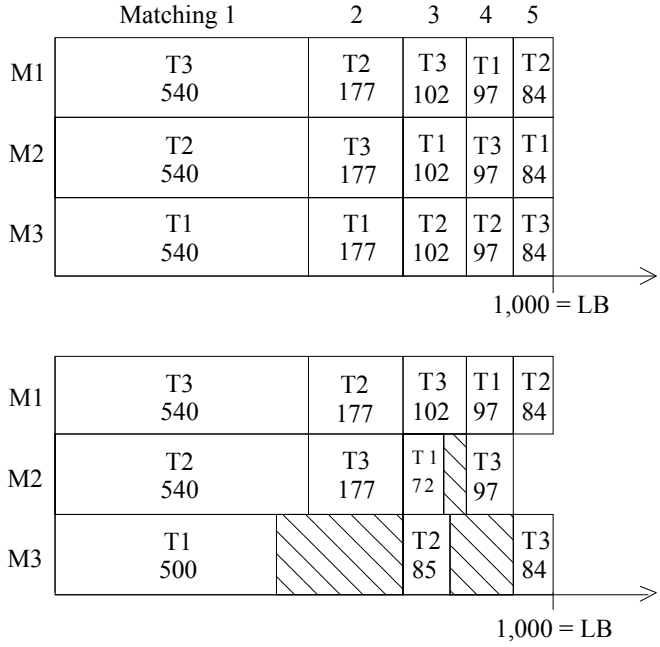


Figure 10.4. *Preemptive scheduling with and without fictitious activity*

The total number of matchings can be reduced by extracting matchings containing a large amount of work. In our example, this goal is achieved by extracting matchings where the smallest element is maximal (*max-min* matching). We can also change the order of matchings in the final breakdown. In this way, by permuting slices 2 and 5 in Figure 10.4, we put back together the two fragments of job 2 on M_1.

10.5. Simple heuristics (excluding metaheuristics)

10.5.1. *Introduction*

Since open shop was studied late by researchers (compared to flow shop and job shop), most published reports first involved complexity results, the study of particular cases, and the demonstration of performance guarantees for certain very simple heuristics. The emergence of methods used in practice for the resolution of

open shop with numerical evaluations over different types of data is very recent. We can cite heuristics from Bräsel *et al.* [BRÄ 93] and from Guéret and Prins [GUE 96]. The following sections present a few results for performance guarantees and two families of heuristics: the list family (priorities) and heuristics-based matching techniques. Current metaheuristics mainly based on the disjunctive model, are the subject of a special section (section 10.7).

10.5.2. *Performance guarantees*

Since open shop is difficult, it is tempting to use approximate methods, but with a deviation to the optimal makespan that is as small as possible. Finding a heuristic which is always at less than 5/4 from the *optimum* is actually an NP-complete problem [WIL 97]. We still do not have a heuristic that is guaranteed at 5/4. We do however have [CHE 93] results for *list heuristics*, which use priority rules to build non-delay schedules (see section 10.2.1). If $C_{max}(H)$ is the makespan calculated by a list heuristic H, then for any instance, $C_{max}(H) / C^*_{max} \leq 2$, i.e. that we are never more than twice the optimum. The bound refines for $m = 3$ machines: 5/3. In addition, Chen defines a rearranging in $O(n)$ for the schedule obtained to decrease the bound to 3/2.

10.5.3. *List heuristics*

Principle

Operations are sorted by decreasing priority in a list Ψ. Each iteration of the heuristic calculates the minimum date t in which operations are executable, and then executes at t the available task with the highest priority.

We obtain Algorithm 10.2, which is very simple if two tables *FM* and *FT* are used. *FM* is a table of m integers defining the date in which each machine becomes available during the algorithm. *FT*, a table of n integers, defines the current end date (completion time) for each job. For each selected operation (u, v), C_{uv} denotes its end date; a matrix C is sufficient to store the schedule obtained without ambiguity.

Examples of traditional priority rules

The following rules are traditional and can apply to open shop:

– *FF (First-Fit)*, where Ψ is the natural order of operations in the instance;

– *Random*, where the list is a random permutation of operations;

– *SPT (Shortest Processing Time)*, operations sorted by increasing times;

– *LPT (Longest Processing Time)*, operations sorted by decreasing times;

– *MWR (Most Work Remaining)*, priority to job with maximum residual time (total time of operations not yet executed).

Initialize FM and FT to 0
for k **from** 1 **to** mn **do**
  $t \leftarrow \min \{ \max(FT(i), FM(j)) \mid (i,j) \in \Psi \}$
  Determine (u, v), 1st operation of Ψ executable at t
  $C_{uv}, FT(u), FM(v) \leftarrow t + p_{uv}$
  Remove (u, v) from Ψ

Algorithm 10.2. *General form of a list algorithm for open shop*

The best rule on average for open shop is *MWR*, followed by *LPT* which is only slightly inferior. Note that *LPT* surpasses *SPT* for open shop, but it is the contrary for flow shop. *FF* and *Random* are interesting because they give an idea of the type of scheduling that a human would manually build. These rules provide a reference solution: an acceptable heuristic must be as good as these two rules. *SPT*, for example, is even worse!

Improved list heuristics

Guéret and Prins [GUE 96] proposed more efficient list heuristics based on vectors of priorities. The priority of an operation (i, j) takes into account its processing time, the duration of its job and the load of its machine. It is in fact a vector (p_{ij}, p_i, Z_j), sorted by component *decreasing order*. In the *H1* heuristic, list Ψ is sorted by a *decreasing lexicographical order* of priorities, just like words in a dictionary: (30, 20, 10) thus has higher priority than (30, 15, 15). *H1* outperforms rule *MWR* on average. An improved heuristic *H1D* is obtained if priorities are managed *dynamically*, i.e. if residual p_i and L_j are updated, followed by priorities after each iteration.

Excellent results are obtained by a heuristic called *H1R*, which is a repetitive version of *H1* accompanied by list perturbations. We begin by applying *H1*. The resulting schedule is examined to detect operations during which gaps appear on other machines and those ending after *LB*. All these operations gain one position in the list. We repeat the process a fixed number of times *NRuns*, while keeping the best schedule found. Algorithm 10.3 clarifies this process.

In a few iterations, a spectacular compaction of the first schedule is often obtained. It is not a local (descent) search method because the makespan of consecutive schedules can increase. Since improvements space out during iterations,

in practice $NRuns = 50$ is a good compromise between quality of results and execution time.

BestCmax ← +∞
Sort Ψ by decreasing lexicographic order
for *Runs* **from** 1 **to** *NRuns* **do**
  Execute list Algorithm 10.2 on Ψ
  //Current solution is kept if it is the best
  if $C_{max} < BestCMax$ **then do**
    $BestCMax \leftarrow C_{max}$
    $BestC \leftarrow C$
  //Priority list perturbation
  for *k* **from** 2 **to** *mn* **do**
    $(i, j) \leftarrow \Psi_k$ (rank *k* operation in Ψ)
    if $C_{ij} > LB$ **or** a machine is inactive in $]t_{ij}, C_{ij}[$ **then**
      Permute Ψ_{k-1} and Ψ_k

Algorithm 10.3. *Repetitive H1R heuristic*

10.5.4. *Matching heuristics*

Principle of matching breakdown

As in the preemptive case, the idea is to extract from *P* matchings corresponding to schedule *slices*, but since we do not start with a quasi-bistochastic matrix, we cannot always find matchings with *m* operations. However, to avoid the generation of too many matchings, we can search for *maximum cardinality* matchings (containing the highest number of non-zero elements). The final schedule corresponds to the concatenation of slices in the order of their extraction. Table 10.10 goes back to our three machines and three job example in which we have chosen a first matching (bordered elements). Figure 10.5 is an example of possible final breakdown.

	M_1	M_2	M_3	p_i
T_1	97	72	500	669
T_2	261	540	85	886
T_3	642	274	84	1,000
L_j	1,000	886	669	***1,000***

Table 10.10. *3 ×3 example with a first matching (bordered)*

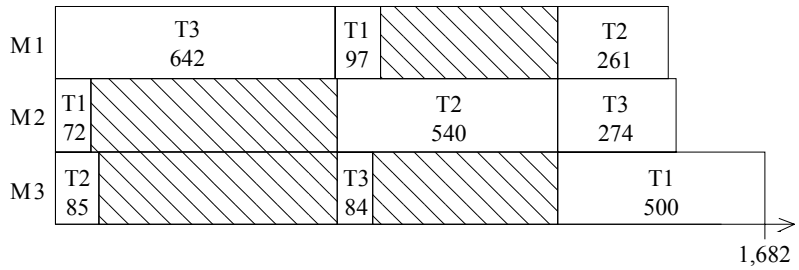


Figure 10.5. *Matching breakdown example for Table 10.10*

It is clear that we can improve the breakdown by "compacting" slices for example, but we can start by searching for a minimal length breakdown. This problem unfortunately is NP-hard, just like open shop, but we can intuitively observe that we will obtain good breakdowns if slices contain a small empty proportion. Here are different types of matchings with the correct property and algorithm references for their calculation (these algorithms are quite technical):

– minimum or maximum sum of processing times (min-weight and max-weight matchings), computable using the "Hungarian" algorithm [PAP 82];

– minimum duration of the longest operation or maximum duration of the shortest operation (min-max and max-min matchings) [DER 78];

– minimum time difference between longest and shortest operations (balanced matchings) [MAR 84];

– *lexicographical* matchings [BUR 91]. *Max-min* example: maximization of the smallest matching element, followed by the second smallest, etc.

Breakdown improvement

Min-max matchings lead to better schedules on average (*H2* heuristic from Guéret and Prins), but even the list heuristic using the *FF* rule is better. The breakdown must therefore be followed by an improvement procedure. We can execute a *compaction* to the earliest operations in each slice, in the order in which they are generated. We can also perform a *compaction-insertion*: before compacting an operation after its predecessor on its machine, we look first for an earlier gap where the operation can be inserted on this machine without conflict. Figure 10.2 shows the schedule obtained by compacting the one from Figure 10.5. A compaction-insertion procedure would give an optimal result lasting $LB = 1{,}000$ by inserting (1, 3) in the gap between (2, 3) and (3, 3) on M_3.

These improvements depend on the slice sequence. Since compaction is quick ($O(mn)$), we can build an initial breakdown (slice sequence), and then carry out a *local search* with the goal of changing the sequence order to reduce makespan after compaction-insertion. For example, at each iteration of this local search, we can explore movements of a slice in the current sequence, or permutations of two slices, and observe if the makespan decreases after compaction-insertion. If that is the case, we execute the transformation and we start again with the new sequence as the current sequence. On average, the best results are obtained by a heuristic called *H2RL*, containing a min-max matching breakdown, followed by a local search with slice movement and permutations, and compaction-insertion [GUE 96].

Theoretical advantage with matching methods over list methods

List methods build *non-delay* schedules and, in some instances, no list will return the *optimum* (see section 10.2.1). That is not the case with matching methods which have more chances of finding the optimum in theory. Decompositions into matchings, either followed by compaction or not, result in *semi-active* schedules, but most often they are not *active* (as in Figures 10.5 and 10.6). On the other hand, compaction-insertion returns active schedules by definition, among which there is always an optimal solution.

10.6. The disjunctive model and shop problems

10.6.1. *Disjunctive model review*

This model is recalled here because it is used by most metaheuristics and exact methods to solve the open shop problem. The disjunctive model [ROY 64] is a graph describing scheduling problems with *sequencing* and *disjunctive constraints*. A disjunctive constraint (or *disjunction*) between two tasks i and j means that i and j cannot occur at the same time. This constraint is most often caused by sharing a resource only available in one copy. More precisely, for this type of n task scheduling problem, the disjunctive graph $G = (X, U, D, P)$ is defined as follows:

– X is a set of *nodes* representing the n tasks as well as two fictitious tasks 0 (input) and $n + 1$ (output);

– U is a set of *arcs* describing sequencing constraints; it must not contain cycles;

– D is a set of *edges* (undirected arcs) representing disjunctions;

– P is a mapping giving for each arc (x, y) the minimum time between the start dates of tasks x and y (generally the processing time p_x of task x).

The advantage of this model is the following: to define a schedule, each disjunction $[x, y]$ must be replaced either by one arc (x, y) (for x before y) or by one

arc (y, x) (for y before x). This decision is called arc selection. Any feasible schedule corresponds to a *complete* selection, in which each disjunction of G is replaced by one arc without creating cycles. For a given complete selection, the makespan corresponds to the longest operation sequence in G, called *critical path*. For a graph with k arcs, this path can be calculated in $O(k)$ by the Bellman algorithm (described in [PRI 94a] and Chapter 2 for example). Several branch-and-bound methods and metaheuristics then become possible by exploring complete selections to find the one with minimal makespan.

10.6.2. *Disjunctive model and shop problems*

For shop problems, the tasks in the disjunctive model correspond to the mn operations. For the flow shop and job shop, the sequencing constraints in U form n paths from fictitious task 0 (graph input) to $mn + 1$ (graph output). These paths correspond to sequences imposed on each job. For the open shop, U is empty, there are only disjunctions. Disjunctions only occur between operations of the same job or the same machine. Complete selections have a simple structure: each operation has two successors and two predecessors (one in its job and one on its machine).

10.6.3. *Example of open shop disjunctive model*

Figure 10.6 reuses the first open shop scheduling example given at the beginning of this chapter. The critical path is easily detectable on the Gantt chart: it is the sequence of operations ((3, 1), (1, 1), (1, 3)) responsible for the makespan 1,239.

Figure 10.7 gives the complete selection of disjunctions corresponding to this schedule. First, we have created mn operation-nodes, and then we have entered the arcs describing the order of operations observed on each machine and in each job. The weight of each arc (x, y) is the processing time of operation x. And finally we have connected the beginning (node 0) to operations with no predecessor by zero time arcs, followed by operations without successors to exiting node $mn + 1$. By applying the Bellman algorithm, we find the critical path and its time (in dotted lines).

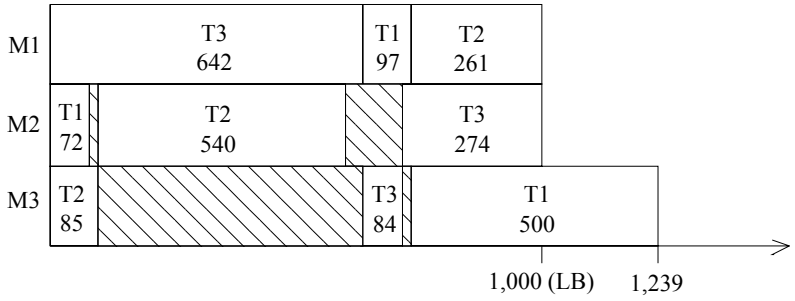


Figure 10.6. *Example of schedule to compute a complete selection*

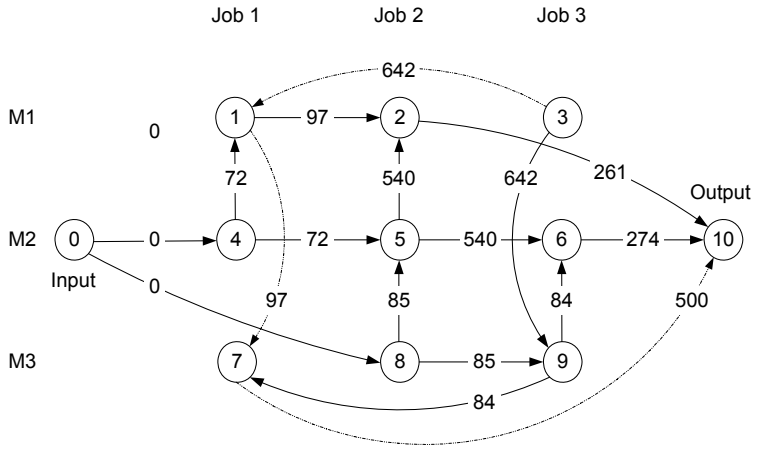


Figure 10.7. *Complete selection of disjunctive graph*

10.6.4. *Disjunctive model properties*

In a complete arbitration, a *block* is a maximum sequence of consecutive operations of the same job or machine, located on the critical path. From the graph structure, a block has at least two operations. A block operation is *internal* if it is not the first, or the last, one of the block. In addition, job-blocks and machine-blocks are *alternated* along the critical path. For example, in the previous example, operations (3, 1) and (1, 1) form a block on machine M_1, whereas (1, 1) and (1, 3) constitute a block for job T_1. We obtain the following properties [BAL 69] for any feasible schedule S of a shop problem by considering the associated complete selection:

– inverting an arc from the critical path results in another feasible schedule;

– inverting a non-critical arc either creates a cycle or does not improve S;

– inverting an arc between two internal block operations does not improve S.

10.7. Metaheuristics for the open shop

10.7.1. *Known traditional neighborhoods for job shop*

Balas properties make it possible to explore a set of potentially improving transformations (*neighborhood*) from a current schedule. They were first used for job shop. The simplest neighborhoods dispense with testing the emergence of cycles, thus their exploration is quick and they have been used for simulated annealing algorithms: inverting an arc from a critical path block [VAN 92], or inverting an arc (x, y) from the critical path, with operation x at the beginning of a block or y at the end of a block [MAT 88]. In [DEL 93], more complex transformations are developed for a tabu search method. They can create cycles: inverting two consecutive arcs in a block, or move an internal operation μ at the head or at the end of a block (in case of cycle, we shift μ by one position to the right or left in its block until we obtain a feasible schedule).

10.7.2. *Tabu search and simulated annealing methods for open shop*

The general principle of these methods is presented in Chapter 3. Three tabu methods were proposed for the open shop. The oldest and simplest one uses the neighborhood applied to job shop from Matsuo [TAI 93]. Alcaide *et al.* then have tested the inversion of one or two consecutive arcs from a block [ALC 97]. The most recent method uses a sophisticated neighborhood [LIA 99b]. For current scheduling and an operation x, Liaw notes as $PJ(x)$ and $SJ(x)$ the predecessor and successor in the job, and $PM(x)$ and $SM(x)$ the predecessor and successor for the machine. For any arc (x,y) of a machine-block, he inverts the three arcs (x, y), $(PJ(y), y)$ and $(x, SJ(x))$. For any arc (x, y) of a job-block, he inverts (x, y), $(PM(y), y)$ and $(x, SM(x))$. The only simulated annealing method was proposed by Liaw [LIA 99a] with the same neighborhood as in his tabu method.

10.7.3. *Population-based algorithms and neural networks*

In 1994, Fang *et al.* proposed the first genetic algorithm for the open shop, with mixed results [FAN 94]. In 2000, Prins developed a flexible genetic algorithm [PRI 00] where the chromosomes are lists of operations. The schedule associated with a chromosome is recovered by three *scheduling engines* which build either a non-

delay schedule (like the list in Algorithm 10.2), or an active schedule (like the Giffler algorithm presented in [FRE 82]). Crossover operators are simple, such as *LOX* (*Linear Order Crossover*), but the algorithm includes several refinements. For example, a few really good heuristic schedules are introduced in the initial population. This population is managed in such a way that each individual has a distinct makespan, and improvement procedures such as the *H1R* heuristic are used as mutation operator. This algorithm made it possible to solve several open instances in literature (see section 10.9.2).

Liaw [LIA 00] proposed another genetic algorithm in 2000, but based on the disjunctive model and hybridized with the local search already used for his tabu search method. In 2005, Colak and Agarwal introduced an original approach based on neural networks [COL 05]. Their method is not more effective than the genetic algorithms of Prins and Liaw, but it is remarkably fast (nearly 10 times faster than the other metaheuristics). The same year, Blum published an effective ant colony algorithm, hybridized with a beam search, called Beam-ACO [BLU 05]. The last breakthrough is a particle swarm optimization algorithm (PSO) designed in 2007 by Sha and Hsu [SHA 07]. The two latter metaheuristics have improved several best-known solutions from a set of hard instances introduced by Guéret and Prins [GUE 99].

10.8. Exact methods for open shop

10.8.1. *Brucker et al. branch-and-bound method*

All current (optimal) exact methods for open shop are branch-and-bound methods based on the disjunctive model. The first effective method comes from Brucker *et al.* [BRU 97]; it succeeds in optimally solving problems up to size 10×10. Only a broad overview is given here, because the details of its implementation are complex. Each node in the search tree contains the disjunctive graph with the disjunctions already selected. At the root, no disjunction is selected yet. A node corresponds to a complete solution if all disjunctions are selected. The *branching* scheme uses a theorem from Brucker which considers the graph G with a complete selection and a critical path μ. He stipulates that if a better complete selection H exists, then a block B of μ and an operation u of B also exist, such that u is located before or after all operations of B in H.

In any node, a matching heuristic which respects disjunctions already selected is executed to obtain a complete selection, lasting UB. For branching, we apply the theorem by choosing a critical path μ of G. Children-nodes are obtained by considering each B block of μ, and by forcing each internal u operation to be before or after any other x operation of B. The generated child-node inherits the

disjunctions selected in the father-node, to which we add arcs (u, x). *Bounding functions* (lower bounds) are calculated. The node is pruned if one of them reaches UB. These bounds are similar to those proposed by Carlier and Pinson [CAR 89] for job shop. They are based on the calculation of a *release date* and *tail* for each operation, from already selected disjunctions. A constraint propagation technique, called immediate selection, deduces additional selections in each node.

10.8.2. *More recent improvements*

Several techniques can decrease the number of nodes explored by the Brucker method by 10. The first method, called *intelligent backtracking* [GUE 00], attempts to identify which selected disjunction is responsible for the pruning of a node S in the search tree. This disjunction may have been selected in a node located several levels above the father-node. It can be detected by keeping some information (*explanation system*) in each node. We can backtrack directly to the selecting node, which accelerates the exploration of the search tree.

Another technique [GUE 98a] is called *prohibited intervals*. Here is its principle for any schedule with makespan UB, a machine M_j with load L_j, and a given operation (i, j) from M_j. In any optimal schedule, M_j cannot contain an idle period higher than $\Delta = UB - L_j$. It is possible to quickly calculate, in $O(n.LB)$, the durations $d_1, d_2, ..., d_k$ of all subsets of operations of M_j not containing (i, j). We can show that (i, j), in an optimal schedule, can only start in intervals such as $[d_u, d_u+\Delta]$, $u = 1, 2, ..., k$. This technique enables the elimination of numerous nodes in which operations occur outside of these intervals. It has actually solved one open 10×10 instance from the literature.

Today, the most effective branch-and-bound algorithm is the one designed by Dorndorf *et al*. [DOR 01]. It combines constraint propagation methods with a technique called *shaving*. This technique consists of considering assumptions such "operation (i,j) starts at its earliest start time" or "operation (i,j) starts at its latest start time" and checking whether a schedule exists under these assumptions. This new method has for the first time solved some 20×20 instances to optimality.

10.9. Algorithm comparison

10.9.1. *Uniform processing times*

The simplest way to test algorithms is to randomly generate examples with uniform processing times drawn in an integer interval $[a, b]$. As with other shop problems, these problems are on average more difficult when $m = n$. Numerical

evaluations in [BRÄ 93] and [GUE 96] with times in [1, 100] show that this type of open shop is actually quite easy: the deviations to *LB* are low, and the optimal makespan actually often equals *LB*. For comparison purposes, this situation is rare in the job shop. Table 10.11 compares heuristics from section 10.5 on examples with times in [1, 100]: list heuristics *FF*, *LPT*, *H1D* and *H1R*, and matching methods *H2* and *H2RL*. For each heuristic, we give the average gap between C_{max} and *LB* (in %), over a series of 1,000 problems and in parentheses, the number of times where *LB* (thus the *optimum*) is reached.

Size	*FF*	*LPT*	*H1D*	*H1R*	*H2*	*H2RL*
5×5	5.23 (201)	4.37 (247)	3.15 (350)	0.43 (778)	8.21 (54)	0.28 (868)
7×7	5.09 (98)	3.95 (133)	3.22 (200)	0.42 (702)	8.49 (8)	0.18 (872)
10×10	4.23 (49)	3.04 (75)	2.61 (81)	0.35 (607)	8.16 (3)	0.12 (883)

Table 10.11. *Comparison between list and matching heuristics*

In this type of example, the decomposition into min-max matchings without improvement procedure (*H2*) returns the worst results, at 8% from the bound. Among list heuristics, even *FF* has a decent 5% gap. However, methods without improvement mechanisms (i.e. all except for *H1R* and *H2RL*) rarely reach *LB* when *n* increases: at most 81 10×10 problems over 1,000 for *H1D*. The repetitive list *H1R* method solves a majority of examples, also with weakening on the large ones. The decomposition into matchings followed by a local search (*H2RL*) returns the best results and is very stable: it reaches *LB* in 87% of cases regardless of size.

H2RL is thus the most powerful simple heuristic, even though it is slower than *H1R*. It improves even more if less dispersed processing times are used, in [50, 100] for example, because it finds more balanced matchings. On the other hand, by increasing dispersal, *H2RL* weakens and is bested by *H1R*. Metaheuristics are even better, albeit more calculation time intensive: the tabu method from Liaw [LIA 99b] and the genetic algorithm from Prins [PRI 00] reach *LB* in 98% of any size examples. As for branch-and-bound methods, they can optimally solve any example of this type up to 10×10 size. Here is an idea of algorithm execution times (programmed in Pascal-Delphi) to treat a 10×10 problem on a 1.8 GHz PC running Windows XP: *H1R* lasts on average 0.02 s, *H2RL* 1 s, the tabu method 6.5 s, the genetic algorithm 13.9 s, and the branch-and-bound methods 1 min.

10.9.2. *Taillard examples*

More difficult tests can be built. In 1993, from amongst thousands of randomly generated examples, Taillard selected a group of 60 examples with greater deviations to *LB* for its tabu search method [TAI 93]. These examples (10 in each size 4×4, 5×5, 7×7, 10×10, 15×15 and 20×20) are stored in J.E. Beasley's "OR Library" at http://people.brunel.ac.uk/~mastjjb/jeb/info.html. The Brucker branch-and-bound method published later resolved all 4×4, 5×5 and 7×7 and six 10×10, for a total of 36 instances. The proven optima exceed *LB* only for some 4×4 and 5×5. The more recent branch-and-bound algorithm by Dorndorf *et al.* [DOR 01] solved all 10×10 instances. Contrary to Brucker's method, it was also applied to the 15×15 and 20×20 instances: all except one 20×20 were solved to optimality, at the expense of significant running times (50 minutes to 5 hours, on a 333 MHz Pentium II PC).

The simple heuristics from section 10.5 greatly deteriorate compared to uniform time examples. Guéret and Prins offset this weakening with the help of two black boxes: *BBL*, which includes different list algorithms including *H1R*, and *BBM* containing matching methods such as *H2RL* [GUE 98b]. These black boxes return the best result of the heuristics that they contain, which compensates for the possible poor performance of a method for some instances. Table 10.12 compares *LPT*, *BBL* and *BBM* with three metaheuristics on the 60 Taillard problems. For metaheuristics, we give the results published for the tabu method by Taillard in 1993 (*TabuT*), for the one from Liaw in 1999 (*TabuL*), and for the genetic algorithm from Prins in 2000 (*GA*). The table shows the average gap to *LB* in %, the gap to the *optimum* when it is known, and the number of problems for which *LB* and known *optima* are reached.

Method	***LPT***	***TabuT***	***BBL***	***BBM***	***GA***	***TabuL***
Gap to LB %	9.02	2.66	2.56	1.70	1.03	0.92
Gap to Opt. %	8.38	2.12	1.89	1.03	0.34	0.23
LB reached	0	0	2	25	33	42
Opt. reached	0	12	3	28	42	43

Table 10.12. *Comparison of methods for the 60 Taillard examples*

The simple heuristic *BBL* and *BBM* black boxes beat Taillard's tabu method, which highlights the weakness of the disjunctive model in the open shop case. To find a majority of *optima*, we must use the disjunctive model with a sophisticated neighborhood (Liaw's tabu search) or Prins' genetic algorithm, not based on the disjunctive model. *TabuL* is globally slightly better, but mostly finds the *optimum*

for 4×4, 5×5 and 7×7, already resolved by Brucker's branch-and-bound method. It only reaches *LB* in 19 of the 30 10×10, 15×15 and 20×20 examples. The genetic algorithm is not as good for small examples but solves 29 out of these 30 large examples. In particular, it breaks one 15×15 and two 20×20. By combining the Brucker method and genetic algorithm, Taillard examples are all defeated, except for one 20×20 where the genetic algorithm finds the best known solution, 1,171, very close to the lower bound *LB* = 1,169.

The two most recent metaheuristics mentioned in section 10.7.3 are able to reach the lower bound for all instances, even for the last 20×20 open instance: the Beam-ACO of Blum [BLU 05] and the PSO from Sha and Hsu [SHA 07]. Therefore, Taillard instances should no longer be used to benchmark open shop algorithms.

10.9.3. *Difficult Brucker and Guéret and Prins tests*

Since the optimal makespan is most often *LB* in the Taillard examples, Brucker *et al*. had the idea to generate 73 even more difficult examples to test their branch-and-bound method in [BRU 97]. With only sizes 3×3 to 9×9, their machine loads and job times are all very close and often equal to *LB*, equal to 1,000 in all instances. Since machines and jobs are critical, the least delay in a schedule prevents reaching *LB*. Despite their small size, these examples are difficult because nine of them have resisted the branch-and-bound method by Brucker: one 7×7, six 8×8 and two 9×9. In addition, of the 64 optimal solutions, 34 exceed *LB* with gaps reaching 18%.

Method	***LPT***	***BBL***	***BBM***	***GA***
Gap to LB %	12.65	5.00	6.83	2.83
Gap to Opt. %	9.60	2.49	4.26	0.45
LB reached	2	10	10	21
Opt. reached	4	16	15	41

Table 10.13. *Comparison of methods for the 73 Brucker examples*

Table 10.13 compares *LPT*, *BBL*, *BBM* and Prins' genetic algorithm with these examples (not used by Liaw and published after Taillard studies). We observe the deterioration of all methods. Processing times are much dispersed in the Brucker examples and make the list methods of *BBL* better than *BBM* matching methods, as mentioned in section 10.9.1. The genetic algorithm is very robust, however, with an average of 0.45% for the 64 known *optima*. The algorithm may reach the *optimum* on some of the nine open problems, but we do not have proof since the *optima*

mainly seem to be higher than *LB* for this type of example, contrary to the examples from Taillard.

Guéret and Prins have found even more difficult instances by designing the first lower bound *NB* better than *LB* for the open shop [GUE 99]. Such bounds have been known for a long time for flow shop and job shop because of the sequencing constraints, but they were lacking with the open shop. The bound is obtained by optimally solving a relaxed problem in which we authorize the simultaneous execution of operations for all jobs, except one fixed job T_k. This problem is still NP-hard but can be efficiently resolved in practice by using the following property: there is a quick algorithm in $O(mn \times LB)$ for the particular case where T_k has an imposed sequence. By generating hundreds of Brucker instances and by calculating the new bound, they have isolated 80 instances 3×3 to 10×10, with *NB*/*LB* reaching 1.3. These 30% gaps are comparable to those of the most difficult job shop cases.

Contrary to Taillard instances, many instances are still open in the two sets described in this section, which can be downloaded from the following Internet address: http://www.emn fr/x-auto/gueret/OpenShop/OpenShop html. Concerning the 73 Brucker instances, the Beam-ACO of Blum has improved six best-known solutions and the very recent PSO of Sha and Hsu two others. However, one 7×7 out of 12, five 8×8 out of 11 and two 9×9 out of 3 are not yet solved to optimality. The situation is even worse for the 80 harder files from Guéret and Prins, even if Blum has improved 24 best-known solution values and Sha and Hsu 16: three 6×6, six 7×7, six 8×8, eight 9×9 and all 10×10 are still undefeated (there are 10 instances for each size). These examples illustrate the hardness of the open shop: in comparison, all flow shop and job shop instances up to 10×10 are easily solved nowadays.

10.10. Open shop problems in satellite telecommunications

10.10.1. *TDMA systems principle*

Modern satellite telecommunications systems (Intelsat, Eutelsat for example) use the *TDMA* (time division multiple access) technique. They raise several scheduling problems presented in a survey paper [PRI 94b]. In these systems, a set of *n Earth-stations* cyclically share over time a *geostationary satellite* to transmit *numerical data packets*. The cycle has a very short time Δ (2 ms). The list of packets to transmit in each cycle is known in advance and does not change for several minutes. We have the time to calculate a packet schedule for a cycle, in order to share the satellite without conflict. This solution is also applied to the following cycles as long as the packets do not change.

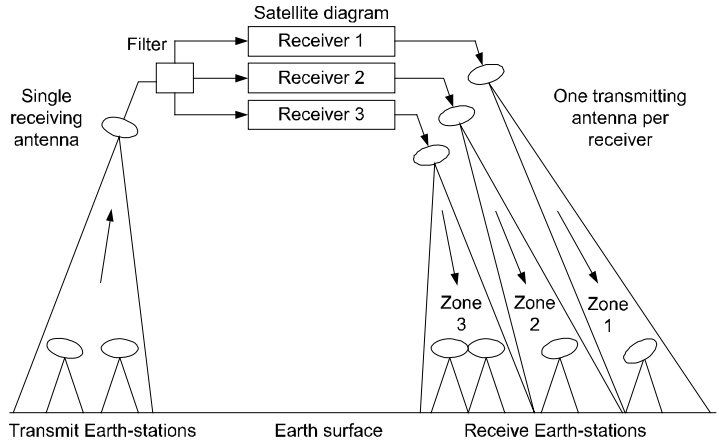


Figure 10.8. *Simplified diagram of a satellite telecommunications system*

The satellite has a single antenna receiving the packets transmitted by the stations (Figure 10.8). This antenna feeds a series of m amplifier tubes (*repeaters*), and each repeater has its own output antenna which covers a *zone* on the ground. Each repeater is sensitive to a specific radio frequency. A filter placed after the receiving antenna switches the packets transmitted by stations to the appropriate repeater according to the frequency. For example, if a station i must transmit a packet to station k located in zone j, this packet $i \rightarrow k$ will have to be transmitted by i to the satellite, over the frequency of the repeater with a transmitting antenna covering k. The satellite will receive this packet, switch automatically according to its frequency to the repeater covering zone j, and the retransmitted packet will thus reach k.

10.10.2. *Pure open shop cases*

In the simplest systems, we have an open shop. In fact, the packets can be assimilated to tasks: since the numerical throughput (bit rate) of the system is constant, we can convert their size, given in bytes for example, into time. Two stations cannot send a packet at the same time to the same repeater, or they risk interference. In addition, a station transmitter (respectively the receiver) has a bus structure which prohibits the transmission (reception) of more than one packet at a time. We can agree that repeaters (or their destination zones) correspond to machines, and traffic transmitted by each, to a job. If a station i has a packet to transmit to a repeater j, this packet can be viewed as a known time operation (i, j). The order of packets is completely open because of the very short system cycle. The

telecommunications protocol is responsible for resequencing the packets upon receipt at the stations with the help of buffers.

The objective is to find a schedule within a cycle time Δ, which can be very difficult if *LB* is close to Δ. In practice, we still attempt to minimize the used part of the cycle to preserve capacity for unexpected packets. Fortunately, this open shop problem is often preemptive: a station can fragment a packet to be transmitted as long as it is transmitted in integers (between two data *bytes*). We can then apply the polynomial Gonzalez and Sahni algorithm from section 10.4.

10.10.3. *Preemptive case complications*

Each packet contains a fixed size preamble. If we decide to cut the packet in two, we must create an additional preamble. Minimizing total time for the preemptive case becomes NP-hard when the preamble size is significant compared to packet size. In certain systems, packets have compressed data. The compression rate decreases for a smaller packet: even without taking the additional preamble into account, splitting a packet increases in a non-linear fashion the volume of data to transmit.

10.10.4. *Generalization of the basic open shop*

We also find problems generalizing the basic open shop in the non-preemptive case. First, we can have several packets with different sizes transmitted by a given station i to a given repeater (or a zone) j: we then have an open shop of the *second type*, in which a job may have several operations per machine. Subsequently, we can have several repeaters in parallel to cover each zone j: this is a *hybrid open shop* case, or *multi-level*. Finally, there can be p stations k_1, k_2, ..., k_p in a zone. Since a packet retransmitted by the satellite in a zone can be picked up by all stations in the zone, we can have *multi-destination packets* in the form of $i \rightarrow k_1, k_2, ..., k_p$ which include traffic for all these destinations. If in addition, we have several repeaters in parallel for the zone, simultaneously transmitting several packets with common destinations is prohibited, since a station cannot receive more than one packet at a time.

The general form can be modeled as a hybrid open shop where station transmitters are jobs and repeaters (or zones) are machines, and where station receivers are *additional renewable resources*. It is extremely difficult in practice, because of the large size and numerous additional constraints in real systems. Carlier and Prins designed methods actually used in the daily use of the Eutelsat European system [CAR 86].

10.11. Conclusion

Open shop problems are increasingly common in production because of flexible shops. We also find them in telecommunications and in timetabling. Timetabling was not addressed, for lack of room, and also because it groups very different problems depending on whether we consider school or university timetables, shift work, or crew or transport system timetables. Certain timetable problems are clearly similar to the problems studied in this chapter, for example the organization of a university exam period: all exams that a student must take can be considered as one open shop job. A review from de Werra provides a good starting point [WER 85].

Long considered as easy in practice compared to flow shop and job shop, the open shop problem is actually characterized by small deviations to the lower bound *LB* for the total time (maximum machine loads and job times), which is often reached. In fact, by digging deep, it is possible to find very difficult tests, with 30% gaps comparable to those of other shop problems. Research still remains to be done for exact methods: a 7×7 example from Brucker remains open, whereas any 10×10 flow shop or job shop example can be optimally resolved today.

The disjunctive model used by branch-and-bound methods and all metaheuristics except for the genetic algorithm by Prins, seems to reach its limits with open shop whereas it is very powerful with job shop. An obvious reason is the absence of sequencing constraints which weaken the model. Another explanation is the useless enumeration of non-active schedules which cannot be optimal (this is the case for the example in Figures 10.7 and 10.8). This is a weakness in the face of a solutions space made huge by free job routes. Fortunately, the genetic algorithm returns very good solutions: on the one hand its intrinsic parallelism adequately scans the solutions space, but on the other hand, it only explores active schedules and is somewhat more “productive”.

10.12. Bibliography

[ACH 82] ACHUGBUE J.O. and CHIN F.Y., “Scheduling the open-shop to minimize mean flow time”, *SIAM Journal on Computing*, vol. 9, p. 306-311, 1982.

[ADI 89] ADIRI I. and AIZIKOWITZ N., “Open shop scheduling with dominated machines”, *Naval Research Logistics*, vol. 36, p. 273-281, 1989.

[ALC 97] ALCAIDE D., SICILIA J. and VIGO D., “Heuristic approaches for the minimum makespan open shop problem”, *Trabajos de Investigación Operativa*, vol. 5, no. 2, p. 283-296, 1997.

[BAL 69] BALAS E., “Machine sequencing via disjunctive graphs: an implicit enumeration algorithm”, *Operations Research*, vol. 17, p. 941-957, 1969.

[BLU 05] BLUM C., "Beam ACO – hybridizing ant colony optimization with beam search: an application to open shop scheduling", *Computers and Operations Research*, vol. 32, p. 1565-1591, 2005.

[BRÄ 93] BRÄSEL H., TAUTENHAHN T. and WERNER F., "Constructive heuristic algorithms for the open-shop problem", *Computing*, vol. 51, p. 95-110, 1993.

[BRU 97] BRUCKER P., HURINK J., JURISCH B. and WÖSTMANN B., "A branch-and-bound algorithm for the open-shop problem", *Discrete Applied Mathematics*, vol. 76, p. 43-49, 1997.

[BUR 91] BURKARD R.E. and RENDL F., "Lexicographic bottleneck problems", *Operations Research Letters*, vol. 10, p. 303-308, 1991.

[CAR 86] CARLIER J. and PRINS C., "Optimisation des plans de trame dans le système Eutelsat AMRT/CNC", *Annales des Télécommunications*, no. 9-10, p. 501-518, 1986.

[CAR 89] CARLIER J. and PINSON E., "An algorithm for solving the job-shop", *Management Science*, vol. 35, no. 2, p. 164-176, 1989.

[CHE 93] CHEN B. and STRUSEVICH V.A., "Approximation algorithms for three machine open-shop scheduling", *ORSA Journal on Computing*, vol. 5, p. 321-326, 1993.

[COL 05] COLAK S. and AGARWAL A., "Non-greedy heuristics and augmented neural networks for the open-shop scheduling problem", *Naval Research Logistics*, vol. 52, no. 7, p. 631-644, 2005.

[DEL 93] DELL'AMICO M. and TRUBIAN M., "Applying tabu search to the job-shop scheduling problem", *Annals of Operations Research*, vol. 41, p. 231-252, 1993.

[DER 78] DERIGS U. and ZIMMERMANN U., "An augmenting path method for solving linear bottleneck assignment problems", *Computing*, vol. 19, p. 285-295, 1978.

[DOR 01] DORNDORF U., PESCH E. and PHAN-HUY T., "Solving the open shop scheduling problem", *Journal of Scheduling*, vol. 4, p. 157-174, 2001.

[DRO 99] DROBOUCHEVITCH I.G. and STRUSEVICH V.A., "A polynomial algorithm for the three-machine open-shop with a bottleneck machine", *Annals of Operations Research*, vol. 92, no. 0, p. 185-210, 1999.

[FAN 94] FANG H-L., ROSS P. and CORNE D., "A promising hybrid GA/Heuristic approach for open-shop scheduling problems", *11th European Conference on Artificial Intelligence (ECAI-94)*, p. 590-594, Wiley, 1994.

[FRE 82] FRENCH S., *Sequencing and Scheduling*, Ellis-Horwood, 1982.

[GON 76] GONZALEZ T. and SAHNI S., "Open shop scheduling to minimize finish time", *Journal of the ACM*, vol. 23, no. 4, p. 665-679, 1976.

[GUE 96] GUÉRET C. and PRINS C., "Classical and new heuristics for the open-shop problem: a computational evaluation", 96/3 AUTO Report, Ecole des Mines de Nantes, published in part in *European Journal of Operational Research*, vol. 107, no. 2, p. 306-314, 1998.

[GUE 98a] GUÉRET C. and PRINS C., Forbidden intervals for open-shop problems, 98/10/AUTO Report, Ecole des Mines de Nantes, 1998. Presented at IFORS-99, Beijing.

[GUE 98b] GUÉRET C. and PRINS C., Efficient heuristic black boxes for the open-shop: comparison with other methods on three benchmarks, 98/11/AUTO Report, Ecole des Mines de Nantes, 1998.

[GUE 99] GUÉRET C. and PRINS C., "A new lower bound for the open-shop problem", *Annals of Operations Research*, vol. 92, p. 165-183, 1999.

[GUE 00] GUÉRET C., JUSSIEN N. and PRINS C., "Using intelligent backtracking to improve branch-and-bound methods: an application to open-shop problems", *European Journal of Operational Research*, vol. 127, no. 2, p. 344-354, 2000.

[LAW 81] LAWLER E.L., LENSTRA J.K. and RINNOOY KAN A., "Minimizing maximum lateness in a two-machine open-shop", *Mathematics of Operations Research*, vol. 6, p. 153-158, 1981.

[LIA 99a] LIAW C.F., "Applying simulated annealing to the open-shop scheduling problem", *IIE Transactions*, vol. 31, no. 5, p. 457-465, 1999.

[LIA 99b] LIAW C.F., "A tabu search algorithm for the open-shop scheduling problems", *Computers and Operations Research*, vol. 26, no. 2, p. 109-126, 1999.

[LIA 00] LIAW C.F., "A hybrid genetic algorithm for the open shop scheduling problem", *European Journal of Operational Research*, vol. 124, no. 1, p. 28-42, 2000.

[MAR 84] MARTELLO S., PULLEYBLANK W.R., TOTH P. and DE WERRA D., "Balanced optimization problems", *Operations Research Letters*, vol. 3, no. 5, p. 275-278, 1984.

[MAT 88] MATSUO H., SUH C.J. and SULLIVAN R.S., A controlled search simulated annealing method for the general job-shop scheduling problem, Working Paper 03-04-88, Graduate School of Business, University of Texas at Austin, 1988.

[PAP 82] PAPADIMITRIOU C.H. and STEIGLITZ K., *Combinatorial Optimization*, Prentice Hall, 1982.

[PIN 95] PINEDO M., *Scheduling: Theory, Algorithms and Systems*, Prentice Hall, 1995.

[PRI 94a] PRINS C., *Algorithmes de graphes*, Eyrolles, 1994.

[PRI 94b] PRINS C., "An overview of scheduling problems arising in satellite communications", *Journal of the Operational Research Society*, vol. 45, no. 6, p. 611-623, 1994.

[PRI 00] PRINS C., "Competitive genetic algorithms for the open-shop scheduling problem", *Mathematical Methods of Operations Research*, vol. 52, no. 3, p. 389-411, 2000.

[ROY 64] ROY B. and SUSSMANN B., Les problèmes d'ordonnancement avec contraintes disjonctives, DS-9 bis note, SEMA, Paris, France, 1964.

[SAH 79] SAHNI S. and CHO Y., "Complexity of scheduling shops with no-wait in process", *Mathematics of Operations Research*, vol. 4, p. 448-457, 1979.

[SEV 94] SEVAST'JANOV S.V., "On some geometric methods in scheduling theory: a survey", *Discrete Applied Mathematics*, vol. 55, p. 59-82, 1994.

[SHA 07] SHA D.Y. and HSU C.Y. “A new particle swarm optimization for the open shop scheduling problem”, *Computers and Operations Research* (available online 27 February 2007).

[TAI 93] TAILLARD E., “Benchmarks for basic scheduling problems”, *European Journal of Operational Research*, vol. 64, p. 278-285, 1993.

[VAN 92] VAN LAARHOVEN P.J.M., AARTS E.H.L. and LENSTRA J.K., “Job shop scheduling by simulated annealing”, *Operations Research*, vol. 40, no. 1, p. 113-125.

[WER 85] DE WERRA D., “An introduction to timetabling”, *European Journal of Operational Research*, vol. 19, p. 151-162, 1985.

[WER 91] DE WERRA D., BLAZEWICZ J. and KUBIAK W., “A preemptive open-shop scheduling problem with one resource”, *Operations Research Letters*, vol. 10, p. 9-15, 1991.

[WER 92] DE WERRA D. and BLAZEWICZ J., “Some preemptive open-shop scheduling problems with a renewable or a non-renewable resource”, *Discrete Applied Mathematics*, vol. 35, p. 205-219, 1992.

[WIL 97] WILLIAMSON D.P., HALL L.A., HOOGEVEEN J.A., HURKENS C.A.J., LENSTRA J.K, SEVAST’JANOV S.V. and SHMOYS D.B., “Short shop schedules”, *Operations Research*, vol. 45, no. 2, p. 288-294, 1997.

Chapter 11

Scheduling Under Flexible Constraints and Uncertain Data: The Fuzzy Approach

11.1. Introduction

Traditional scheduling problem formulations can be classified into two approaches: those based on the optimization of criteria such as total duration, and those based on the satisfaction of constraints pertaining to delay or resource availability. The first approach calculates an optimal schedule that is not always useful in practice because other criteria may have been neglected in the model. In the second approach, we can formalize local specifications, but we run the risk of either finding no solution if the problem is too constrained, or finding too many and not be able to guide the user, whose preferences are not represented. In addition, most scheduling models are deterministic and assume a precise knowledge of data such as task execution times, production requirements, etc. There is literature on stochastic scheduling, but these models may be difficult to use in practice, as scheduling problems in a deterministic environment are already very complex. Moreover, the stochastic approach [HER 05] often works "in the average", which may not make much sense for production processes or projects that are not repeatedly performed a sufficient number of times.

Using fuzzy sets and possibility theory can make it possible to propose an acceptable trade-off between expressivity and computational difficulty, when modeling preferences and uncertainty, even though this type of approach is not yet widely used in scheduling. Fuzzy PERT studies have highlighted the difference

---

Chapter written by Didier DUBOIS, Hélène FARGIER and Philippe FORTEMPS.

between a flexible constraint problem and a problem with uncertain execution times. In the first case, we focus on the notion of solution optimality. In the second case, we encounter problems in adapting the critical path method. The flexible constraint approach offers a trade-off avoiding rigidities of constraint analysis while being less restrictive than the optimization of global criteria regarding the representation of objectives. Extending constraint analysis, it also includes certain optimization problems as particular cases. In addition, possibility theory also offers a natural framework; simpler and less greedy in information than the stochastic approach to treat uncertainty for certain parameters such as uncontrollable times for certain tasks when this uncertainty is caused by a lack of knowledge. Flexible constraints and uncertain data can therefore be represented in a single framework. Instead of optimizing behaviors on average, we are searching for schedules that can withstand hazards by satisfying all constraints to a reasonable extent with a sufficient level of confidence.

The use of fuzzy sets in scheduling dates back to the end of the 1970s [DUB 78, PRA 79]. We find reviews on this subject in the book by Loostma [LOO 97] and papers by [CHA 98, WER 99, TUR 99, DUB 03a]. Different studies were published, emphasizing constraint flexibility, ill-known data or the use of fuzzy rules. For example, we can cite:

– project scheduling, having to deal with renewable and non-renewable resource constraints, with multiple modes of task execution and with multiple criteria. It is a pretty realistic and general model [BLA 86]. Fuzzy sets are used to model partial information (uncertainty) on task execution times [HAP 94];

– scheduling based on heuristic rules attempts to express preferences between task sequences in a constructive manner. In this perspective, the literature shows the application of fuzzy heuristic rules to scheduling problems with known execution times on one hand [BEL 88], and in a more general way the use of fuzzy priority rules [GRA 94];

– job-shop problem under imprecise execution times, addressed on the basis of methods for comparing fuzzy quantities, and resolved by simulated annealing type methods [FOR 97b];

– job-shop problem with flexible constraints, for which we try to minimize job due dates by minimizing violation of the most violated constraint. Similar to project scheduling, this problem comes down to setting hard constraints by reaching a compromise between local criteria in order to ensure the existence of a small number of solutions at the “center” of the original constraint domain [FAR 97];

– the hybrid case where there are preferences about due-dates and execution times for certain tasks, and where such times for other tasks are uncertain. We

demonstrate that this problem comes down to the pure flexible constraints case when we look for robust solutions [DUB 95a].

This chapter is intended to be an introduction to these studies. We will first present a refresher on possibility theory and fuzzy intervals. Section 11.3 follows with a discussion on the case of scheduling under flexible constraints with fuzzy priority rules. Section 11.4 describes the fuzzy approach with ill-known execution times for activities. We examine difficulties encountered by the critical path method under uncertainty and provide recent solutions to these difficulties. We also show how to formulate the problem of task sequence determination under uncertainty. Finally, the last section before the conclusion briefly discusses the hybrid case where fuzzy constraints and uncertain execution times are present simultaneously.

11.2. Basic notions on the fuzzy approach to uncertainty and constraints

11.2.1. *Possibility theory*

In order to model unknown or flexible quantities, we must recall a few results from fuzzy sets and possibility theory (see for example [DUB 88]). A fuzzy set A is a subset of a universe U whose boundaries are gradual. Formally, the following definitions are instrumental.

The *membership function* μ_A of fuzzy set A associates with each element $u \in U$ a degree of membership $\mu_A(u)$ from u to A, normally with values in [0, 1].

The *core of A* is the set $c(A) = \{u, \mu_A(u) = 1\}$. It gathers elements totally belonging to A (they are prototypes of A).

The *height* of a fuzzy set A is the maximum degree of membership of an element from U to A. If this height is smaller than 1, the core is empty.

The *level-cut* α of A is the *crisp* set (i.e. non-fuzzy) A_α of elements of U with a degree of membership to A of at least α. These level-cuts form a family of nested sets and constitute another representation of the fuzzy set A.

The *support* of A is the set $s(A) = \{u, \mu_A(u) > 0\}$. It contains prototypes of A and peripheral elements (borderline elements such as $0 < \mu_A(u) < 1$). It only rejects "*non-A*" prototypes ($\mu_A(u) = 0$).

The *complement* of a fuzzy set A in universe U is denoted A^c and its membership function is $\mu_{A^c} = 1 - \mu_A$. Fuzzy set *union* and *intersection* are obtained by respectively using the pointwise maximum and minimum of membership functions.

Given a parameter x with an ill-known value, all that we know is that value x belongs to a fuzzy set A: the stronger the degree of membership $\mu_A(u)$ of a value u to fuzzy set A, the more it is possible (i.e. plausible) for u to be the value of x. Values outside the support are considered impossible. In other words, the value of x is restricted by a *possibility distribution* $\pi_x = \mu_A$. We assume that the core of A is not empty, i.e. $\mu_A(u) = 1$ for at least one value u.

In the context of this type of uncertainty modeling, a possibility distribution is similar to a distribution of probability. It is different due to the conventions adopted: $\pi_x(u) = 1$ does not mean that $x = u$ is certain, but only that it is a plausible value. In particular, the sum of degrees of possibility can be greater than 1. In terms of modeling, possibility degrees do not account for the same uncertain aspect as probabilities. Possibility theory describes the incomplete character of information (possibly total ignorance when for any u, $\pi_x(u) = 1$) whereas probabilities often express randomness. Note that these representations are not incompatible: incompleteness in probability is captured by probability intervals, and each possibility distribution efficiently, and very simply, encodes a specific family of probability measures. In fact, probability and possibility are only incompatible in the context of a Bayesian approach that mandates that the probability distribution be unique.

The possibility of an event "$x \in P$", denoted by $\Pi(x \in P)$, is the intersection degree of A and P, where the intersection of two fuzzy sets is defined by the minimum operation:

$$\Pi(x \in P) = \sup_u \min (\mu_A(u), \mu_P(u)) \quad [11.1]$$

This degree assesses to what extent event "$x \in P$" may be true or, in a similar way, to what extent proposition "$x \in P$" is consistent with information "$x \in A$" modeled by $\pi_x = \mu_A$. Note that in the previous equation, P can also be a fuzzy set.

The dual *necessity measure* of "$x \in P$", denoted as $N(x \in P)$ evaluates to what extent A is completely included in the core of P. In other words, to what extent proposition "$x \in P$" is certainly true, i.e. that it is implied by information "$x \in A$":

$$N(x \in P) = \inf_u \max (1 - \mu_A(u), \mu_P(u)) = 1 - \Pi(x \in P^c) \quad [11.2]$$

where P^c is the complement of P. In fact, $N(x \in P) = 1$ if and only if the support of A is included in the core of P: $x \in P$ is certain if and only if *all* partially possible values of x are among those completely satisfying proposition $x \in P$.

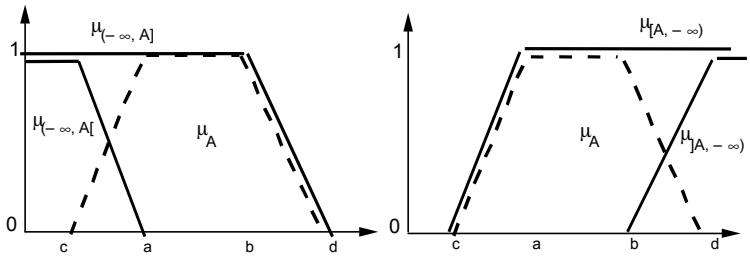


Figure 11.1. *(a) Numbers possibly/necessarily before A; (b) numbers possibly/necessarily after A*

A *fuzzy interval* is a fuzzy subset of reals whose level-cuts are intervals (generally closed). The simplest representation of a fuzzy interval uses a trapezoidal membership function (Figure 11.1): we set two nested intervals *[a, b] ⊆ [c, d]* respectively forming the core and support of the fuzzy interval and we linearly interpolate between *a* and *c*, and between *b* and *d*, respectively.

If *x* is a true value variable, *A* a fuzzy interval and *p* a traditional number, we can verify that:

$$\Pi(x \geq p) = \Pi(x \in [p, +\infty)) = \sup_{u \geq p} \mu_A(u) = \mu_{(-\infty, A]}(p) \quad [11.3]$$

$$N(x \geq p) = N(x \in [p, +\infty)) = \inf_{u < p} (1 - \mu_A(u)) = \mu_{(-\infty, A[}(p) \quad [11.4]$$

$$\Pi(x \leq p) = \Pi(x \in (-\infty, p]) = \sup_{u \leq p} \mu_A(u) = \mu_{[A, +\infty)}(p) \quad [11.5]$$

$$N(x \leq p) = N(x \in (-\infty, p]) = \inf_{u > p} (1 - \mu_A(u)) = \mu_{]A, +\infty)}(p) \quad [11.6]$$

where *(-∞, A], (-∞, A[, [A,+ ∞),]A,+ ∞)* indicate sets with numbers possibly before A, necessarily before A, possibly after *A* and necessarily after *A* respectively (Figure 11.1).

11.2.2. *Fuzzy arithmetic*

We review here some basics on the calculation of fuzzy intervals. *A* and *B* are two fuzzy intervals restricting possible values of two logically independent variables *x* and *y*. The sum and difference between two intervals is defined by:

$$\mu_{A \oplus B}(z) = \sup_{x, y \ z = x + y} \min(\mu_A(x), \mu_B(y)) = \sup_x \min(\mu_A(x), \mu_B(z - x)) \quad [11.7]$$

$$\mu_{A \ominus B}(z) = \sup_{x,\, y \;\; z = x - y} \min(\mu_A(x), \mu_B(y)) = \sup_x \min(\mu_A(x), \mu_B(z + x)) \quad [11.8]$$

When $B = \{b\}$ is a precise value, we find $\mu_{A \ominus B}(z) = \mu_A(z - b)$ and $(-\infty, A] \oplus b = (-\infty, A \oplus b]$.

Similarly, the minimum and maximum of two fuzzy intervals are fuzzy intervals defined by (see also Figure 11.2):

$$\mu_{\tilde{\min}(A,B)}(z) = \sup_{x,\, y \;\; z = \min(x,\, y)} \min(\mu_A(x), \mu_B(y)) \quad [11.9]$$

$$\mu_{\tilde{\max}(A,B)}(z) = \sup_{x,\, y \;\; z = \max(x,\, y)} \min(\mu_A(x), \mu_B(y)) \quad [11.10]$$

Extended minimum and maximum operations satisfy most usual minimum and maximum properties, for example idempotence, or property $\tilde{\max}(A, B) \oplus C = \tilde{\max}(A \oplus C, B \oplus C)$, or even mutual distribution. However, as Figure 11.2 shows, the $\tilde{\max}(A, B)$ fuzzy interval cannot be equal to A or to B, as well as for fuzzy interval $\tilde{\min}(A, B)$.

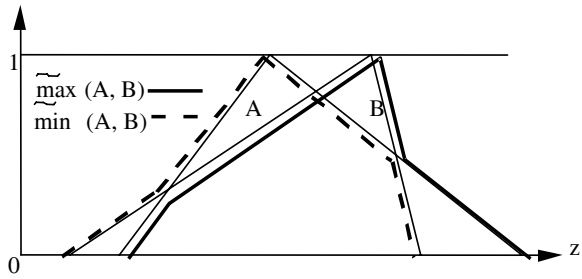


Figure 11.2. *Maximum and minimum of two fuzzy intervals A and B*

11.2.3. *Fuzzy interval comparison*

Fuzzy interval comparison is a tricky problem as shown by the large amount of literature involved. Numerous techniques have been developed, and each one was inspired by a different semantic. We are giving a broad overview by choosing the relevant methods in the present framework. For more information, please refer to [BOR 85, CHE 92, DUB 00, WAN 01].

First, we must distinguish between two situations: the case where fuzzy intervals must actually be totally ordered (for example, the comparison of total fuzzy execution times for two different schedules) and the case where we must describe the relative position of fuzzy intervals with the help of a degree of comparison (the

degree to which an inequality holds, for example). If the goal is to determine the degree of satisfaction of constraint $A \geq B$ between two flexible quantities A and B, i.e. encoding preferences formulated in the form of fuzzy intervals, we use formulae defining the possibility that some x smaller than A and larger than B exists for example [DUB 88]:

$$\Pi(A \geq B) = \sup_x \min(\mu_{(-\infty, A]}(x), \mu_{[B, +\infty)}(x))$$

This purely metric index is the direct extension of the comparison of intervals and is based on the following idea: $[a, b] \geq [c, d]$ if and only if $a \geq d$. In this case, $\Pi([c, d] \geq [a, b]) = 0$ and $\Pi([a, b] \geq [c, d]) = 1$. If $[a, b] \cap [c, d] \neq \varnothing$, then $\Pi([c, d] \geq [a, b]) = \Pi([a, b] \geq [c, d]) = 1$, which indicates indifference. The properties of this index make it the natural possible extension of interval orders [FOD 94, PIR 97].

On the other hand, when ranking imprecise quantities, we give preference to a method based on calculation of areas with an interpretation in terms of imprecise probabilities. In short, the method called "area compensation" [FOR 96] belongs to a family of "defuzzification" approaches consisting of replacing fuzzy intervals by real substitutes and comparing them. The real or true substitute $F(A)$ of a fuzzy interval A is defined as the center of gravity of the average interval generated by the family of probabilities compatible with fuzzy interval A [YAG 81, DUB 87]. Mathematically, we can define the real substitute of A by:

$$F(A) = \frac{1}{2}\int_0^1 (a_\alpha^- + a_\alpha^+) d\alpha$$

where $[a_\alpha^-, a_\alpha^+]$ is the level cut α of A. There are other fuzzy interval defuzzification methods. Nevertheless, contrary to other methods, our proposition has the advantage of linearity, i.e. it guarantees: $F(A \oplus B) = F(A) + F(B)$ and $F(\lambda \cdot A) = \lambda F(A)$, for any scalar λ.

11.2.4. *Possibilistic utility*

Possibility theory offers a framework different from the one of expected utility for decision under uncertainty. If we stick to the traditional context of decision theory in which a decision is a function δ from a set of states S to a set of consequences X, we assume that knowledge of the state of a system is described by a possibility distribution π over S and that preference in the decision result is modeled by a function $\mu: X \rightarrow [0, 1]$ such that $\mu(x)$ is the degree of preference of consequence

x (similar to a utility function). We then have two criteria based on degrees of necessity and possibility [11.2] and [11.1], to evaluate a decision δ:

– pessimistic criterion: $u_*(\delta) = \inf_{s \in S} \max (1 - \pi(s), \mu(\delta(s)))$;

– optimistic criterion: $u^*(\delta) = \sup_{s \in S} \min (\pi(s), \mu(\delta(s)))$.

By considering that a state s such that $\pi(s) = 1$ is seen as a plausible state, $u_*(\delta)$ evaluates to what extent decision δ presents favorable consequences in all plausible situations. In particular if we have no information on the state, then $\pi(s) = 1 \ \forall s$ and $u_*(\delta) = \inf_{s \in S} \mu(\delta(s))$; we find Wald's pessimistic criteria. On the other hand, $u^*(\delta)$ is high when a plausible state exists where decision δ leads to a successful consequence. These criteria are different from the expected utility based on a probability *Prob* (such as $u(\delta) = \Sigma_S Prob(s) \cdot \mu(d(s))$, for two reasons:

– they assume less information on the state;

– they apply to decisions which are not repeated, where results do not accumulate (contrary to costs for example).

The rationality of these criteria was highlighted by their axiomatizations in the style of Von Neumann and Morgenstern [DUB 95b] and Savage [DUB 01b] respectively.

11.3. Scheduling under flexible constraints

The use of flexible constraints in the context of scheduling (job-shops, for example, but also in project management) is a way to offset deficiencies with traditional scheduling approaches. The search for an *optimum* in terms of a unique criterion does not interpret well the fact that the reasons why one schedule is better than another often depend on the satisfaction of local constraints. Although the constraint-based approach (see Chapter 5) actually tries to take this local aspect into consideration, it is based on an all-or-nothing form, whereas approaches using optimization evaluate the quality of a schedule in a more gradual way. Using flexible constraints enables us to remain gradual while maintaining the essence of the constraint-based approach, which excludes compensation between local criteria. First, we treat the main scheduling problem (with unlimited resources), followed by sequencing problems caused by limited resources.

11.3.1. *The fuzzy PERT problem: flexible constraints*

We consider sets of tasks linked together by precedence constraints, indicating that a task cannot start before the other one ends. The set forms what we call a project or a job. By convention, two fictitious tasks with zero duration time are added, α (start of the project) and ω (end of the project); α precedes all tasks, and all tasks precede ω. When resource constraints are not considered, a project can be represented by a directed graph with no cycle where the nodes represent activities and arcs show precedence constraints. *Succ(i)* (resp. *Pred(i)*) represents all tasks immediately following (resp. preceding) *i*, whereas *SUCC(i)* (resp. *PRED(i)*) represents all successors (resp. predecessors) of *i*. Given two tasks *i* and *j* of the graph, *C(i, j)* represents all paths ϕ from *i* to *j*. We mainly use notations from Chapter 2. We also note by t^-_j and t^+_j the earliest and latest starting times for task *j*.

The scheduling problem considered here is a matter of determining task starting times and to possibly set their execution times *on the basis of availability and delivery constraints*. In this case, a delivery date *d* and/or an availability date *r* are specified for the project, and we can use the following constraints in addition to precedence constraints between tasks (remember that initial and final tasks have zero times):

$$\textit{Availability constraint: } t_\alpha \geq r \qquad [11.11]$$

$$\textit{Delivery constraint: } t_\omega \leq d \qquad [11.12]$$

This constraint satisfaction problem may very well fail to have a solution (choice of task starting times t_j). The earliest starting time of each task only depends on the duration of tasks preceding it and the availability date, whereas the latest starting date depends on the delivery date, the duration of this task and the duration of tasks following it. If the earliest and latest starting times of the task are incompatible, then the problem has no solution. In this case, negative floats can emerge for certain tasks. However, it is quite possible that no task is critical.

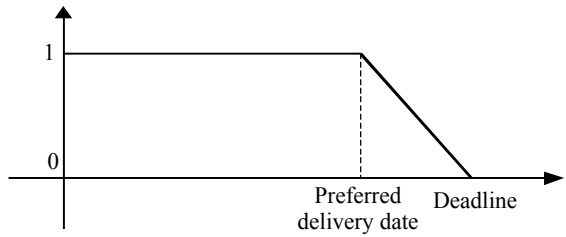


Figure 11.3. *Flexible delivery date*

Even though certain tasks have to comply with local availability (waiting for parts before continuing for example) or delivery constraints (milestone dates for example), there are additional constraints which must be taken into consideration, in the form of a local feasibility window:

$$r_i \leq t_i \leq d_i - p_i \qquad [11.13]$$

We must admit that these constraints are often "flexible" in practice. We can make the temporal window where a product must be manufactured flexibly, by modeling supplier and customer preferences. Customers wish to receive finished products before a certain date, called preferred delivery date; but they may accept some delay. Generally, there is a maximum deadline beyond which the order will be considered obsolete: either the parts ordered will no longer be useful or another supplier is used. Between the preferred delivery date and the deadline, customer satisfaction decreases. The longer the delay, the less satisfied the customer will be.

This type of decreasing satisfaction curve, represented by a fuzzy interval $(-\infty, \tilde{d}]$ in Figure 11.3, can only be obtained during negotiation with the customer. A global flexible constraint on delivery results in a local satisfaction index $sat(t_\omega) = \mu_{(-\infty,\tilde{d}]}(t_\omega)$.

We formalize in a similar way supplier preferences. In short, they prefer to supply as late as possible to ensure a certain temporal float for the manufacturing and delivery of semi-finished products. A global flexible constraint on availability is satisfactory at level $sat(t_\alpha) = \mu_{[\tilde{r},+\infty)}(t_\alpha)$. The fuzzy interval $[\tilde{r}, +\infty)$ has an increasing membership function such that $\mu_{[\tilde{r},+\infty)}(x) = 0$ if $x \leq \underline{r}$ (impossible to start before $\underline{r}$) and $\mu_{[\tilde{r},+\infty)}(x) = 1$ if $x \geq \bar{r}$ (prefer to start after $\bar{r}$).

Finally, task execution times can sometimes be controlled and are subject to preferences, for example by setting machine parameters (speed, instrument precision, temperature of a tank, etc.), or by allocating more or less manpower (e.g. preferences depend on the number of employees to pay). The criteria involved thus affect the quality of item produced, and the comfort or safety of operators. For each task, we assume that there are optimal parameter settings, and acceptable, even if less satisfactory, parameter values (not as close to security, quality and wear standards). We can represent the preference relative to task duration by a fuzzy interval. In other words, the execution time of a task i can be a controllable parameter whose more or less satisfactory values can also be modeled by a fuzzy interval $\tilde{p}_i$: a duration p_i of task i instantiation is totally satisfying if it is part of optimal duration values, i.e. to the core of $\tilde{p}_i$. It is not at all satisfying (it violates

the constraint) if it is outside the range of acceptable values, i.e. support of $\widetilde{p}_i$. Otherwise, the closer it gets to the set of optimal values, the more satisfactory it becomes. In other words, we have a flexible constraint for p_i where the degree of satisfaction by an instantiation p_i is the degree of membership of p_i to $\widetilde{p}_i$: $sat(p_i) = \mu_{\widetilde{p}_i}(p_i)$.

Generally, the "useful" part of fuzzy interval $\widetilde{p}_i$ is only its increasing part that contradicts due-date constraints. It can sometimes interpret the following preference: the more time allocated to the task, the better it is (because the result is of a better quality, the job is done under better conditions, etc.).

To summarize, we consider these different preferences in the choice of starting and execution times for tasks to be scheduled. Consequently, we look for a *plan* that will better satisfy contradictory constraints of suppliers, customer and resources used to accomplish the tasks.

In this context, a potential solution to the problem is an instantiation $(t_\alpha, \ldots, t_\omega, p_\alpha, \ldots, p_\omega)$ of the starting times of *n* tasks and their execution times that must satisfy "traditional" hard constraints, such as precedence constraints between tasks on one hand, and on the other hand flexible availability, delivery and duration constraints described above. We must therefore solve a problem of fuzzy constraint satisfaction [DUB 95a, DUB 96]: the degree of satisfaction of a potential solution $(t_\alpha, \ldots, t_\omega, p_\alpha, \ldots, p_\omega)$ is the degree of satisfaction of the least satisfactory constraint. $sat(t_\alpha, \ldots, t_\omega, p_\alpha, \ldots, p_\omega) = 0$ if $(t_\alpha, \ldots, t_\omega, p_\alpha, \ldots, p_\omega)$ violates at least one precedence constraint, and otherwise:

$$sat(t_\alpha, \ldots, t_\omega, p_\alpha, \ldots, p_\omega) = \min\left(\min_i \mu_{\widetilde{p}_i}(p_i),\ \mu_{(-\infty, \widetilde{d}]}(t_\omega),\ \mu_{[\widetilde{r}, +\infty)}(t_\alpha)\right) \quad [11.14]$$

We have not taken into consideration possible local delivery or availability constraints here, relative to specific tasks. We can observe that potential solutions are not equally satisfying. Degrees of satisfaction lead to a total pre-order of all instantiations that satisfy hard constraints. A solution violating a date or a constraint on duration is unacceptable (*sat* = 0); whereas there are solutions that satisfy all ideal task duration and starting time ranges, they are the best (*sat* = 1); otherwise, the problem becomes conflicting and we must explicitly accept a relaxation of preferences on execution times. The goal is to stay as close as possible to ideal execution times.

The degree of coherence *cons* of the problem (degree of satisfaction for the best solutions) somewhat expresses the degree of feasibility of the scheduling problem:

$$cons = \sup_{(t_\alpha,\ldots,t_\omega\,,\,p_\alpha,\ldots,\,p_\omega)} sat(t_\alpha,\ldots,t_\omega,p_\alpha,\ldots,p_\omega) \quad [11.15]$$

When constraints are partially inconsistent, we find $0 < cons < 1$, as no instantiation can satisfy them perfectly.

In practice, the problem comes down to the search for starting time instantiations $(t_\alpha,\ldots,t_\omega)$, by merging precedence constraints and flexible constraints for execution times. Let $(t_\alpha,\ldots,t_\omega)$ be an instantiation of starting times only; it is satisfactory when there exists $(p_\alpha,\ldots,p_\omega)$ such that $(t_\alpha,\ldots,t_\omega,p_\alpha,\ldots,p_\omega)$ optimizes the problem:

$$sat(t_\alpha,\ldots,t_\omega) = \sup_{(p_\alpha,\ldots,p_\omega)} sat(t_\alpha,\ldots,t_\omega,p_\alpha,\ldots,p_\omega) \quad [11.16]$$

We can show that:

$$sat(t_\alpha,\ldots,t_\omega) = \min_{(j,k)\ \text{such that}\ j\in Pred(k)} \min(\mu_{[\tilde{p}_j,+\infty)}(t_k - t_j)\ ,\mu_{(-\infty,\tilde{d}]}(t_\omega),\mu_{(-\infty,\tilde{d}]}(t_\omega)) \quad [11.17]$$

Thus, we have to solve a problem only involving task starting times, containing two flexible availability and delivery constraints and *flexible precedence constraints* in the form:

$$t_j - t_i \in [\tilde{p}_i,+\infty) \quad [11.18]$$

with a degree of satisfaction equal to $\mu_{[\tilde{p}_i,+\infty)}(t_j - t_i)$.

Instead of directly searching for a solution, the user is generally interested in knowing the set of possible starting time values for each task. A value for the starting time of task *i* is satisfactory insofar as it is higher than its earliest starting time and less than its latest starting time caused by the set of precedence constraints, availability and delivery. It is obvious that if global availability and delivery dates are decision variables of the problem, the earliest and latest task starting times are independent groups of variables. This reasoning is still valid in the flexible case, because we can reason with availability and delivery dates as well as flexible execution times as if they were not fuzzy. We can in fact calculate the degree of local satisfaction $sat(t_i)$ of an instantiation of the earliest starting time t_i of task *i:*

$$sat(t_i) = \sup_{(t_j:\, j\neq i)} sat(t_\alpha,\ldots,t_\omega) = \min(\mu_{[\tilde{t}^-_i,+\infty)}(t_i),\mu_{(-\infty,\tilde{t}^+_i]}(t_i)) \quad [11.19]$$

where fuzzy earliest and latest starting times of task i are calculated as in the non-fuzzy case because of independent forward and backward propagation steps with the help of operations on fuzzy intervals, and no longer on traditional numbers (by means of addition, subtraction minimum and maximum of fuzzy intervals).

$$\tilde{t}^{-}_{i} = \max_{j \in Pred(i)} \tilde{t}^{-}_{j} \oplus \tilde{p}_j \quad \text{for } i > \alpha, \text{ and } \tilde{t}_{\alpha} = \tilde{r} \qquad [11.20]$$

$$\tilde{t}^{+}_{i} = \min_{j \in Succ(i)} \tilde{t}^{+}_{j} \Theta \tilde{p}_i \quad \text{for } i < \omega, \text{ and } \tilde{t}^{+}_{\omega} = \tilde{d} \qquad [11.21]$$

The more or less satisfactory values for the starting time of task i is therefore a fuzzy interval $[\tilde{t}^{-}_{i}, \tilde{t}^{+}_{i}]$. The local degree of satisfaction $sat(t_i)$ is the height of this fuzzy subset. The degree of coherence of the problem can be known immediately after the forward propagation phase: it is the height of the fuzzy interval $[\tilde{t}^{-}_{\omega}, \tilde{d}]$, from which the project preferred ending date is derived.

If $cons < 1$, we can thus calculate precise values for starting times and task execution times (otherwise, we obtain intervals). Task execution times are obtained by inverting the membership function of the increasing part of $\tilde{p}_i$.

Earliest starting time values are obtained similarly from $\tilde{t}^{-}_{i}$. Nevertheless, if the solution obtained is optimal in the sense of the *bottleneck* type criterion defined by formula [11.14], it is not always Pareto-optimal in the sense of the multicriteria problem involving membership functions of flexible constraints, viewed as criteria to maximize. In fact, only duration and starting times of critical tasks (those whose local degree of satisfaction $sat(t_i)$ coincide with global degree of satisfaction *cons*) are calculated correctly. They form a set of critical paths in the usual sense. We can still improve task execution times along non-critical paths. In order to do this, we can solve a second fuzzy PERT problem, where we set critical task execution times based on $p_i = \mu^{-1}_{\tilde{p}i}(cons)$ and starting and ending project dates likewise. We proceed recursively in this way until all tasks in the graph become critical [DUB 99] (if $cons < 1$).

EXAMPLE.– A set of three tasks $T1$, $T2$, $T3$ where $T1$ must precede $T2$ and $T3$. We must find earliest starting times and task execution times. Each task is assumed to last 4 hours to be properly completed. There is no way to complete it in less than two hours. We note as (a, b, c) the fuzzy number of core b and support (a, c). The feasibility domain of p_1 is $(2, 4, +\infty)$. Task $T1$ can start at time $r = 0$. Task $T2$ must be preferably ended within four hours and $T3$ within six hours. Tasks must be finished within eight hours in any case. Fuzzy delivery dates are therefore $(-\infty, 4, 8)$

and $(-\infty, 6, 8)$. Let t_1, t_2, t_3 be task starting times, p_1, p_2, p_3 their best execution times. The degree of coherence of the problem is the maximum value of:

$$\min(\mu_{(2,4,+\infty)}(p_1), \mu_{(2,4,+\infty)}(p_2), \mu_{(2,4,+\infty)}(p_3), \mu_{(-\infty,4,8)}(t_2 + p_2), \mu_{(-\infty,6,8)}(t_3 + p_3))$$

with $t_1 \geq 0$, $\min(t_2, t_3) \geq t_1 + p_1$. We can confirm that $cons = 0.5$. However, the value selection of execution times and starting times of degree of membership 0.5 gives $p_1 = p_2 = p_3 = 3$, and suggests $t_1 = 0$, $t_2 = 3$, $t_3 = 3$. This solution can be improved.

The set of tasks $T1$, $T2$ is critical. It cannot be moved without decreasing *cons* below 0.5 (this value enforces an ending date of 6). Variables t_1, t_2, p_1, p_2 are completely determined on the critical path. However, $T3$ is not critical in this configuration. It will be executed in the best conditions by optimizing the simpler scheduling problem with variables t_3 and p_3, defined by: $t_3 \geq 3$, $t_3 + p_3 \in (-\infty, 6, 8)$, and $p_3 \in (2, 4, +\infty)$. A simple calculation shows that the optimal max-min solution is unique and gives $t_3 = 3$ and $p_3 = 3.5$ with a degree 0.75. Joined to values found previously for t_1, t_2, p_1, p_2, it forms an optimal Pareto as well as optimal max-min solution for the problem.

11.3.2. *Limited resources: flexible constraints and fuzzy rules*

The practical realization of a project is greatly constrained by the availability of resources. We limit ourselves here to resources known as "renewable", i.e. resources that are integrally restored at the end of execution of the task requesting them. They are most often machines, but also additional tools or even human operators. Often disjunctive, these constraints prevent the execution of more than one task at a time even when no priority precedences between these tasks exist.

The flexible constraints approach easily extends to resource constraint scheduling, a special case of which is job-shop scheduling. In this case, resources are machines. A task requires a specific machine during its execution. This particular case is very important in reality because it involves a significant part of job-shop models.

Next to temporal parameters, other constraints can be subject to flexibility, notably the use of resources. If we have generally interchangeable machines, we can express preferences on the use of one machine or the other to execute a task. Outside of traditional machine type resources, we can also consider human resources. If we typically have a certain quantity of operators, it may become necessary to hire additional personnel in order to deal with the workload in a reasonable amount of time. This increase in human resources constitutes a relaxation of the initial constraint; it is accompanied by additional cost as well as a possible deterioration of

production quality. It may occur that the additional employees are less efficient because they are less specialized. Finally, in contexts such as open shop (see Chapter 10), precedence constraints can also be flexible. Namely, even if there is a preference on the precedence between two tasks A and B on the same product, the other precedence is tolerable. For example, we can hope that A precedes B (with a degree equal to 1), while accepting that B can precede A with a positive degree less than 1.

The resolution procedure for the job-shop problem under flexible constraints is derived from fuzzy PERT under flexible constraints. In fact, we can apply the general principle of changing resource constraints into (disjunctions of) precedence constraints. Namely, any job-shop scheduling problem under flexible constraints can be seen as a family of flexible constraint project scheduling problems, in which we must find the best representative – in terms of max-min criteria.

Methods have been developed and experimented for the resolution of job-shop scheduling problems under flexible constraints [FAR 97]. They are implicit enumeration techniques, where the search in the solution graph, notably to solve conflicts of precedence between tasks, is guided by degrees of membership of current partial solutions. We have been able to show that taking into consideration flexibility of constraints does not slow down the solving process. On the contrary, the heuristic associated with problems involving preferences tends to position tasks in the middle of temporal windows, often making it possible to reduce calculation time.

Determining task sequences on machines was addressed by some with the help of fuzzy rules that model heuristic knowledge in a flexible way and express a blending of priority rules. The use of priority rules in fact has the advantage of easily solving the sequencing problem, possibly in real time, even if the objectives attained with local priority rules are not always very clear. In most production scheduling software packages, heuristics, such as empirical rules, are used, making it possible to rank tasks according to a particular priority. This ranking is then used in a scheduler. A simulation tool can gradually allocate the necessary resources to the different tasks, while respecting precedence constraints. When a conflict between tasks appears because of resource sharing, the program may select the next priority task according to the heuristic involved. Among traditional priority rules, we can name SPT (Shortest Processing Time) and LST (Least Slack Time) that rank tasks according to their execution times and estimated floats respectively (see Chapter 6). In fact, a great number of priority rules were proposed for project management, and in a more general way, for production scheduling [MAC 93]. The use of fuzzy sets in priority rules enables two improvements. First, with a fuzzy priority rule, we avoid thresholding effects. We can also make a blending of the different traditional rules.

For example, it is reasonable to write: *if task duration is small and if its float is very small, then the task must receive high priority*. This linguistic rule naturally translates in terms of fuzzy rules:

If duration ∈ SMALL *and* float ∈ VERY SMALL *then* priority ∈ HIGH

where "SMALL" "VERY SMALL" and "HIGH" are fuzzy intervals, for example.

When we need to settle a conflict between two tasks requiring the same machine, we match the parameters of these tasks against the premises of the rule to determine an indication of their respective relevance. The actual crisp duration and float values are used to determine the activation degrees of the premises, the degree of activation of a "high priority" conclusion is calculated here as the minimum of activation degree of rule conditions. We avoid defining a crisp partition between situations where a rule applies and other situations, although similar to the previous ones, where another longer applies.

The combination of several rules in a fuzzy expert system makes it possible to improve the advice provided by this system. Since the different priority rules influence the performance in different ways according to prescribed criteria that measure scheduling quality, their mode of combination does not constitute a trivial problem. For example, it is well known that if the criterion considered is average cycle time, SPT and LST rules are not equally relevant. In this case, SPT is more appropriate and should therefore receive higher weight in the decision process.

In [BEL 89], the authors jointly use the approach by constraints and fuzzy priority rules combined with a majority approach for job-shop scheduling. The aggregation of different fuzzy priority rules was studied by [DUB 94], and the difficult problem of choosing weights by [GRA 94], in the context of general scheduling problems. Ideal weights are based on the number of resources, dispersion of execution times and obviously the criteria to be optimized. That is why additional rules can be added to the rule-based system for determining priority weights of fuzzy rules, for example the fuzzy rule:

if the dispersion of operation times is high and if minimizing average delay and average cycle time are significant, then the SPT weight must be high.

The implementation details of a fuzzy rule-based system are out of the scope of this chapter (see [DUB 88] for more information). The four main steps for the inference process are:

– *pattern matching*: linguistic terms (e.g. high, important) are compared with real input values (e.g. operation duration) to determine the degree of truth of each premise;

– *inference*: premise compatibility values are propagated (according to a generalized *modus ponens*) toward conclusions;

– *combination:* conclusions of the different rules are combined to obtain a global conclusion of the rule base;

– *selection:* a crisp conclusion is calculated from the fuzzy conclusion.

[TUR 99] proposes an extensive bibliography on the fuzzy rule-based approach for planning and scheduling, an approach that Turksen himself largely developed and applied in the artificial intelligence context.

11.4. Scheduling with ill-known execution times

In this second version of fuzzy scheduling, we try to minimize total completion time (or makespan) of a set of tasks. We no longer have constraints expressing preference on dates or execution times. As with their traditional counterparts, that they generalize, fuzzy scheduling techniques under uncertainty require that the problem considered can be represented in the form of a directed, compact and acyclic graph. This model is convenient for the efficient processing of time windows and execution times, fuzzy or crisp, as well as conjunctive precedence constraints.

We assume task execution times are non-controllable parameters, i.e. the user is not the one choosing their values. They are, for example, subcontracted production management task execution times, software debugging tasks in computer system project management, process execution times that depend on the resource that will eventually be assigned to tasks, etc. As a consequence, the values of execution times p_i for tasks i are no longer known with precision and certainty: we only know more or less plausible values in the form of fuzzy sets $\widetilde{p}_i$. A particular case is that of execution times for which we only know a bracketing interval. While the idea of representing ill-known execution times by intervals sounds very natural, it seems paradoxical that the interval PERT problem was considered and solved only recently.

Noting that, in traditional PERT, the values of all relevant parameters come from calculations based on task execution times, we may be tempted to apply the same algorithm as with the deterministic case to PERT, by performing calculations on intervals or fuzzy intervals as in the previous section instead of calculations on traditional numbers. In fact, the critical path method seems difficult to directly adapt, and most previous fuzzy PERT methods handing uncertain durations are *ad hoc* in this respect. This type of problem requires a "potential-bound" type representation ([ESQ 95]). When, in addition, we want to take limited resources into

account, appropriate precise representative values of uncertain execution times are used when solving the sequencing problem.

11.4.1. *Critical paths under ill-known execution times: difficulties*

The necessity for a relevant model representing uncertainties in execution times in the manufacturing area no longer needs to be proven. By nature, these execution times are subject to hazards and are sometimes only partially controllable. In particular, if we consider infrequent tasks or new types of operation resulting from a technological evolution, using information-demanding methods for determining the value or the statistical distribution of these execution times is not relevant. These considerations remain valid in project management, notably when tasks are still ill-defined.

Modeling these imprecise execution times using fuzzy intervals alleviates these drawbacks by exploiting simple and usually available pieces of information [ROM 90, FOR 97b]. It is rarely possible to obtain a complete description of fuzzy intervals representing imprecise execution times. It is in fact illusory to require perfect precision on data encoding uncertainty. In this field, recommendations from Rommelfanger [ROM 90] can be instrumental in practice. To obtain a fuzzy interval, he proposes obtaining from the decision-maker (or the expert) a few characteristic values and interpolating in a linear way between these points: which values are typical (core)? Which values are not so surprising (level cut 0.5)? What are the boundaries of the physical domain (support)? The corresponding fuzzy interval is then obtained by linear interpolation. Fuzzy intervals for execution times are encoded with the help of such six specific values.

We consider the traditional problem of project duration minimization. In this case, we have no availability or delivery constraint; the objective is to minimize total project duration. This usually leads to posing the following constraints on earliest and latest starting times:

$$t^-_\alpha = 0 \text{ (by convention, we consider that the project starts at date 0)} \quad [11.22]$$

$$t^-_\omega = t^+_\omega \text{ (we consider that the project ends as soon as possible)} \quad [11.23]$$

In this problem, we try to define possible ranges for choosing relative task starting times. The difference between the latest starting time and earliest starting time of a task (called total float) represents its leeway for starting it without delaying the end of the project. Since this type of problem always has a solution, the floats are never negative.

When execution times p_i have precise values, we can consider them as constants. Both problems, minimization of project duration and availability and delivery constraint scheduling respectively, are resolved in a similar manner. A first phase of *forward propagation* (in the style of equation [11.20]) calculates task earliest starting times from $t^-_\alpha = 0$ (resp. $t^-_\alpha = r$), and a *backward propagation* phase (in the style of equation [11.21]) calculates task latest starting times from $t^+_\omega = t^-_\omega$ (resp. $t^+_\omega = d$, since task ω has a duration of zero). The floats can thus be calculated as $\tau_i = t^+_i - t^-_i$ and critical tasks are (possibly) detected. These are zero float tasks: we cannot delay them without delaying the end of the project (see Chapter 2).

When execution times p_i have ill-known values, the most natural thing is to calculate the fuzzy set $\tilde{t}^-_i$ of generally possible values of earliest starting times by forward propagation, iteratively applying formula [11.20] with fuzzy execution times. We are then tempted to pose $\tilde{t}^+_\omega = \tilde{t}^-_\omega$, and to execute a *backward propagation* by iteratively applying formula [11.21].

As highlighted in [CHA 81, DUB 88, NAS 93, ROM 94, HAP 94], the forward propagation phase is correct, in the sense that the calculated $\tilde{t}^-_i$ do correspond to all generally possible values for earliest starting times. A difficulty appears with backward propagation when we wish to respect the constraint $t^+_\omega = t^-_\omega$. Since the earliest starting time t^-_ω of the last task is ill-known and is represented by an uncertainty distribution $\tilde{t}^-_\omega$, requiring that the distribution of t^+_ω be the same as that of t^-_ω is not the same as prescribing that $t^+_\omega = t^-_\omega$, because two quantities can at the same time be independent and have the same distribution. For example, consider a very simple one-task problem. We have the following dependencies between quantities:

$$t^-_1 = t^-_\alpha = 0; \qquad t^-_\omega = t^-_1 + p_1; \qquad t^+_\omega = t^-_\omega; \qquad t^+_1 = t^+_\omega - p_1 \ (= t^-_1)$$

Since quantities t^+_ω and p_1 are not logically independent (because of constraints $t^+_\omega = t^-_\omega = t^-_1 + p_1$), it is not correct to write that all possible t^+_1 values can be obtained by subtraction of fuzzy intervals $\tilde{t}^+_\omega = \tilde{t}^-_\omega$ and $\tilde{p}_1$. For example, if the set of generally possible values for p_1 is interval [1, 4], we obtain for $\tilde{t}^+_\omega = \tilde{t}^-_\omega$ interval [1, 4], which would yield for $\tilde{t}^+_1$ interval $[1,4] \ominus [1,4] = [-3,3]$, whereas it is obvious that the task is critical and that $t^+_1 = t^-_1$.

More generally, when we pose $t^+_\omega = t^-_\omega$, quantity t^+_ω depends on parameters p_i. Since the t^+_j depend on t^+_ω, these quantities thus depend on p_i. We cannot use the

backward propagation formula $t^+_i = \min_{j \in Succ(i)} (t^+_j - p_j)$ via simple fuzzy interval arithmetics [11.21][1].

In fact, since $A \oplus B \ominus B \neq A$ (more precisely, $A \subset A \oplus B \ominus B$, in the sense of the inclusion of fuzzy sets), applying the backward propagation phase as defined by formula [11.21] runs the risk of considering duration uncertainties twice, which results in fuzzy sets that are too large to actually be useful.

11.4.2. *Critical paths with interval execution times*

The simplest way of representing ill-known duration times is to use intervals of the form $\tilde{P}_i = [p^-_i, p^+_i]$. For a more precise approach to the problem, we must go back to its mathematical representation. For a project with n tasks, we denote by *configuration* a tuple of n values $\Omega = (v_1, ..., v_n)$ such that $v_i \in \tilde{P}_i$. Ω represents the state of the world in which it turns out that $p_1 = v_1$, $p_2 = v_2 \ldots$ and $p_n = v_n$. For a given configuration Ω, $p_i(\Omega)$ represents the value of p_i in Ω. In this configuration, we can thus calculate $t^-_i(\Omega)$, $t^+_i(\Omega)$ and $\tau_i(\Omega)$ which are the earliest and latest starting times and float i in configuration Ω. If ϕ is a path in the graph relating task i to task j, $l^{\Omega}(\phi)$ represents the length of this path for configuration Ω. The set of configurations is equated to the Cartesian product $H = \times_{i=1,n} \tilde{P}_i$. An important role is played by *extreme configurations* such that $p_i(\Omega) \in \{p^-_i, p^+_i\}, \forall i=1, ..., n$.

A configuration defines an instance of a deterministic scheduling problem (classical PERT/CPM problem), on which the PERT/CPM method can be applied. Using configurations, the sets of possible values $t^-_i(\Omega)$ for the earliest starting date and the possible values $t^+_i(\Omega)$ for the latest starting dates and the possible values $\tau_i(\Omega)$ for the float are defined by

$$T^-_i = \{t^-_i(\Omega) \mid \Omega \in H\} = [t_i^{--}, t_i^{-+}]; \; T^+_i = \{t^+_i(\Omega) \mid \Omega \in H\} = [t_i^{+-}, t_i^{++}];$$

$$F_i = \{\tau_i(\Omega) \mid \Omega \in H\} = [\tau_i^-, \tau_i^+].$$

Since functions that define the earliest starting times, latest starting times and floats in terms of task durations are obviously continuous, the quantities T^-_i, T^+_i, F_i,

---

1 Note that during the forward propagation phase, we also perform calculations on non-independent values. This is, for example, the case where two parallel tasks have a common predecessor: their two earliest dates depend on the duration of this common predecessor. When they are immediately followed by a single task, we then calculate the maximum between sets of values of two non-independent variables. However, the hidden dependence problem is avoided since $\max(\tilde{a} \oplus \tilde{b}, \tilde{a} \oplus \tilde{c}) = \tilde{a} \oplus \max(\tilde{b}, \tilde{c})$.

are closed intervals. Chanas *et al.* [CHA 02b] propose to extend the notion of criticality to interval-valued scheduling problems as follows:

– a task is *possibly critical* if there exists a configuration $\Omega \in$ H in which this task is critical in the usual sense;

– a task is *necessarily critical* if it is critical in the usual sense in all configurations $\Omega \in$ H.

Similar definitions can be stated for possibly and necessarily critical paths. While possibly critical paths always exist, necessarily critical paths may fail to exist. When all execution times are interval-valued, there is at most one necessarily critical path. When there is none, there may nevertheless exist isolated (groups of) necessarily critical tasks (for instance, the first and last tasks are always so). Finding them is crucial but turns out to be harder than in the deterministic case, since we can no longer rely on critical paths. It can be proved that a task is possibly critical if and only if the lower bound of its float interval is zero, and that it is necessarily critical if and only if the upper bound of its float interval is zero. If a task is necessarily critical then it has the same earliest and latest starting times intervals but the opposite is not true. This last proposition makes it clear that the float interval of a task cannot be calculated by means of its earliest and latest starting times intervals. It must be calculated separately; see Dubois *et al.* [DUB 05] for details.

The latest starting time of task i in configuration $\Omega \in$ H is defined as the difference $t^+_i(\Omega) = max_{\phi \in C(\alpha, \omega)} l^{\Omega}(\phi) - max_{\phi \in C(i, \omega)} l^{\Omega}(\phi)$ between the longest path in the graph and the longest path between task i and the end-task. Calculating the interval of latest starting times comes down to applying interval analysis methods to this function, which is non-trivial since the two parts of this expression have some ill-known quantities in common. A brute-force method with exponential complexity consists of calculating $t^+_i(\Omega)$ for all extreme configurations consisting of assigning to each task execution time its upper or its lower bound, and calculating the maximum and minimum of $t^+_i(\Omega)$ on such configurations. A simplification consists of noting that, to calculate the lower (respectively upper) bound of $t^+_i(\Omega)$, it is always possible to set the execution time of tasks not succeeding i to their lower (respectively upper) bounds [DUB 03b]. An efficient path enumeration algorithm for determining the bounds of the latest starting time interval is proposed in [DUB 05], based on considerations on the form of optimal extreme configurations. For example, there exists a path $\phi \in C(i, \omega)$ such that an extreme configuration Ω defined as $p_j(\Omega) = p_j$ for j outside ϕ, and p^+_j otherwise, minimizes $t^+_i(\Omega)$. Zielinski [ZIEL 05] proposes a polynomial iterative labeling method for constructing optimal configurations minimizing and maximizing $t^+_i(\Omega)$.

The calculations of float intervals of tasks must be carried out independently, applying interval analysis to the explicit expression of the float, i.e.:

$$\tau_i(\Omega) = = max_{\phi \in C(\alpha, \omega)} l^{\Omega}(\phi) - max_{\phi \in C(i, \omega)} l^{\Omega}(\phi) - max_{\phi \in C(\alpha, i)} l^{\Omega}(\phi).$$

Again, a brute force method can be used, enumerating extreme configurations, and efficient path enumeration techniques can be used to find the optimal bounds of the float intervals [DUB 05]. However, Chanas and Zielinski [CHA 02a] proved that the intrinsic complexity of proving the possible criticality of a task is exponential, hence calculating the greatest lower bound of float intervals of tasks is NP-hard. Surprisingly, the problem of deciding the necessary criticality of a task was solved more recently and turns out to be a polynomial problem [FOR 05] to which a variant of Zielinski's labeling polynomial method can be applied to construct an optimal configuration minimizing the upper bound of the float of a task.

Finally, it was shown that, if the task graph is series-parallel, the interval PERT problem can be resolved by an exact, not very combinatorial, method, based on the simple identification of configurations maximizing or minimizing the quantities of interest (earliest, latest dates and floats) [FAR 00, DUB 05].

11.4.3. *Critical paths with fuzzy execution times*

In order to obtain a more expressive representation of incomplete knowledge about task execution times p_i, a possibility distribution $\pi_i = \mu_{\tilde{p}i}$ can be used. In other words, the degree of possibility that p_i is equal to a given value v is $\Pi(p_i = v) = \pi_i(v) = \mu_{\tilde{p}_i}(v)$. With the hypothesis that task execution times are logically independent, the degree of possibility of a configuration is:

$$\Pi(\Omega) = \Pi(p_1 = p_1(\Omega) \text{ and } \ldots \text{ and } p_i = p_i(\Omega) \text{ and} \ldots \text{ and } p_n = p_n(\Omega)) \quad [11.24]$$

$$= \min_{i=1,n} \Pi(p_i = p_i(\Omega)) = \min_{i=1,n} \mu_{\tilde{p}_i}(p_i(\Omega))$$

The calculation of the possibility or necessity of a criticality event, such as for example the possibility that a path be critical or that a task be critical, or the calculation of possible value distribution quantities defined according to execution times (latest dates and floats in particular) must refer to this joint possibility distribution. The possibility that a path $C(\alpha, \omega)$ going from initial node α to end-node ω of the graph is critical is the possibility of having a configuration in which $C(\alpha, \omega)$ is critical:

$$\Pi(C(\alpha,\omega) \text{ critical}) = \sup_{\Omega: C(\alpha,\omega) \text{ critical in } \Omega} \pi(\Omega) \quad [11.25]$$

Chanas and Zielinski [CHA 01] propose a method to find degrees of possible criticality of tasks. Similarly, the necessity that a path $C(\alpha, \omega)$ be critical is [CHA 02b]:

$$N(C(\alpha, \omega) \text{ critical}) = 1 - \Pi(C(\alpha, \omega) \text{ non-critical}) = 1 - \sup_{\Omega: C(\alpha,\omega) \text{ non-critical in } \Omega} \pi(\Omega) \quad [11.26]$$

denoting G_λ the directed graph where task execution times are represented by level-cuts λ of fuzzy intervals. Since a task is critical if and only if it belongs to a critical path, the degree of possibility that it is critical is the maximum degree of possible criticality of a path from α to ω containing it:

$$\Pi(C(\alpha,\omega) \text{ critical }) = \sup \{\lambda: C(\alpha,\omega) \text{ possibly critical in } G_\lambda\}.$$

The necessity that it be critical is defined by duality.

$$N(C(\alpha,\omega) \text{ critical }) = 1 - \inf \{\lambda: C(\alpha,\omega) \text{ critical in } G_\lambda\}.$$

Concerning the calculation of latest fuzzy dates, and possibly floats, a first solution is to carry out a symbolic calculation as proposed by [NAS 93], operating simplifications on symbolic expressions of latest starting dates and floats. Nevertheless, this symbolic approach is very time-consuming and does not scale up. Another, less calculation intensive, approach is to change the definition of the fuzzy subtraction [ROM 94, HAP 94] so as to make it better adapted to the backward propagation technique for latest starting times. This type of approach is unfortunately *ad hoc* and does not provide the correct fuzzy bounds for latest starting times and floats. Such correct fuzzy bounds consist of applying the extension principle to the explicit expressions of these quantities, without resorting to fuzzy arithmetic. Using a mathematical representation of fuzzy intervals as pairs of (left and right) profiles, a direct extension to fuzzy intervals of the path enumeration algorithms in [DUB 05] was proposed by Fortin [FOR 06].

11.4.4. *Limited resources: approach by fuzzy interval comparison*

We know that the resolution of project scheduling problems with limited resources (such as the job-shop problem) generally consists of searching for the "best" transformation of disjunctive constraints into precedence constraints; the optimization implied by the term "best" defines a possible underlying multi-criteria optimization problem. This transformation thus reduces the problem to simple PERT scheduling, which can be solved by simple critical path techniques in the deterministic case. In the case of ill-known data, it leads to solving a fuzzy or

interval PERT type problem at each step of its resolution [BLA 86, FOR 97a]. Special cases of resource-constrained scheduling problems with ill-known execution times, such as flow-shops, have also been considered, as in [MCC 92].

The operations research literature is replete with methods solving deterministic scheduling problems with limited resources, notably for job-shops. Those recognized as the most efficient result from the joint use of a disjunctive graph and an optimization meta-heuristic (simulated annealing for example), in the deterministic [VAN 92] as well as fuzzy context [FOR 97b]. The idea is simple: if we represent a job-shop type problem to be solved using a disjunctive graph, each eligible solution requires the choice of a specific direction for edges connecting tasks requiring the same resource. Starting from any such solution, we can reach an optimal one (minimizing the makespan), by gradually modifying the direction adopted for certain edges on the longest path in this graph. Local search methods, such as simulated annealing or tabu search for example, favoring modifications that improve performance with regards to criteria chosen, are likely to end up with a very good quality solution, or even an optimal one, in a reasonable amount of time.

Concerning the uncertain duration variant of the problem, it is clearly more relevant to quickly search for a good solution than to persist in trying to determine the optimal solution. In fact, data is in essence imprecise. In a scheduling context with limited resources, ill-known temporal parameters are clearly liable to be modeled by fuzzy quantities, but it is all the same with renewable and countable resources. Thus, when the resource considered consists of a team of operators, their number can also be partially unknown (because of partially uncontrollable availability).

Optimization is performed with the goal of determining the best schedule in the sense of performance criteria such as total duration for example. In other words, we determine the task sequence that leads to the earliest total completion time for each machine. Since task execution times are ill-known, a method that ranks fuzzy intervals is needed. The sequence of tasks on each machine is then clearly determined even if there unavoidably remains imprecision about their starting and ending times. The fuzzy interval representation of imprecise statistical information for temporal parameters naturally leads to the choice of the area compensation method as the proper tool to compare two schedules on the basis of their fuzzy evaluation for example, e.g. their total fuzzy duration (see [FOR 97] and section 11.2.3). Chanas and Kasperski [CHA 03, CHA 04] propose an optimization method directly handling uncertain execution times in specific one-machine scheduling problems. Kasperski [KS 05] applies the same approach to a sequencing problem. An application example of this general scheduling model with resource constraints with fuzzy execution times is described in [HAP 94]. It is a software design tool based on an object-oriented approach. It involves scheduling the different design

steps and therefore the evaluation of the total necessary completion time for creating a piece of software. As an illustration, [HAP 94] presents a specific problem involving a total of 53 tasks (the last one for integrating the different modules and testing the system).

11.5. Flexible constraint scheduling and ill-known task execution times

In the case when there are flexible constraints on due-dates (earliest starting times, latest end dates, delivery dates), but where task execution times are uncertain, we are facing a decision problem under uncertainty. We must choose starting times for tasks (and therefore their sequencing on each machine) in order to satisfy local flexible constraints, regardless of the actual task execution times in their interval of uncertainty. In this case, the problem can be addressed as follows [DUB 95a]. Consider a set of partially sequenced tasks. The fulfillment of each task *i* requires an ill-known processing or execution time $\tilde{p}_i$ as in section 11.4. There is a date at which the project can start, and a due-date. These dates, $\tilde{r}$ and $\tilde{d}$ respectively, are flexible, as in section 11.3.

Suppose that a solution sets the starting times of all tasks. A precedence constraint relating two tasks *i* and *k*, will be more or less certainly verified because of the limited knowledge about execution times. The corresponding degree of necessity that *i* precedes *k* is (applying equation [11.2]):

$$N\,(p_i \leq t_k - t_i) = \mu_{]\tilde{p}_i,+\infty)}\,(t_k - t_i), \qquad [11.27]$$

where $]\tilde{p}_i, +\infty)$ has an increasing membership function, corresponding to the fuzzy complement of the right part (decreasing) of $\tilde{p}_i$. This degree of necessity is smaller as the interval $[t_i, t_k]$ is small, since the smaller the interval, the harder it is to insert task *i*.

The degree of satisfaction of the flexible release constraint of product *j* is simply $\mu_{[\tilde{r},+\infty)}\,(t_\alpha)$, t_α being the starting time of the first task. The degree of satisfaction of the flexible delivery constraint is the degree of certainty that the project terminates on time. If t_k is the starting time of a task *k* immediately preceding the last task ω, we calculate, based on [11.2]:

$$N(t_k + p_k \leq \tilde{d}\,) = \inf_x \max\,(1 - \mu_{\tilde{p}_k}(x)\,,\ \mu_{(-\infty,\tilde{d}]}(x + t_k)\,) \qquad [11.28]$$

We recognize in [11.28] the pessimistic possibilistic preference functional introduced in section 11.2.4, because if $\pi = \mu_{\tilde{p}_k}$ is the characteristic function of an interval $[a, b]$, we have:

$$N(t_k + p_k \leq \tilde{d}) = \inf_{x \in [a, b]} \mu_{\tilde{p}_k}(t + x) = \mu_{\tilde{p}_k}(t + b),$$

which evaluates the satisfaction of this flexible constraint in the worst case.

Overall, the scheduling problem is addressed here as one of maximizing the degree of satisfaction of the most violated constraint among those described above (precedence, release and delivery):

$$\min(\min\{\mu_{]\tilde{p}_i, +\infty)}(t_k - t_i): i, k \in succ(i)\},\ \mu_{[\tilde{r}, +\infty)}(t_\alpha),$$
$$\min\{N(t_k + p_k \leq \tilde{d}): k \in prec(\omega)\})$$

An optimal solution to the problem is therefore a choice of starting times t_i for tasks i maximizing this criterion.

Consider the following small example, with only one task (a trip), the starting date of which we must calculate [DUB 95]. Consider a person living in the suburbs of a city and traveling downtown by bus to an important meeting. The meeting is scheduled to take place at 8 am. We create the following constraints:

– the person prefers not to leave home before 7 am, at least not before 6:30 am. We then have a constraint $\tilde{r}$ on the starting time, such that $\mu_{[\tilde{r}, +\infty)}(t) = 0$ if $t \leq$ 6:30 am and 1 if $t \geq 7$;

– the person does not want to arrive at the meeting after 8:15 am. Ideally, (s)he wishes to arrive at 8 am. Therefore, there is a constraint $\tilde{d}$ on the arrival time, such that $\mu_{(-\infty, \tilde{d}]}(t + p) = 1$ if $t + p \leq 8$ and 0 if $t + p \geq$ 8:15.

– the travel duration p is uncertain as the bus may arrive right away at the bus stop, or there will be a waiting time depending on traffic density. We have a possibility distribution $\pi = \mu_{\tilde{p}}$ such that $\pi(p) = 1$ if $p = 1$ and $\pi(p) = 0$ if $p \notin]45,$ 1:15[. Travel time is therefore approximately one hour and we have no control over it.

The optimal start hour is obtained by maximizing:

$$\min(\mu_{[\tilde{r}, +\infty)}(t), N(t + p \leq \tilde{d})), \qquad [11.29]$$

where $N(t + p \leq \tilde{d}) = \inf_p \max(1 - \pi(p), \mu_{(-\infty,\tilde{t}]}(t + p))$. This last expression is a pessimistic possibilistic criterion. We observe that calculation of the optimal value is simplified if we note that, under continuity conditions on membership functions,

$$\inf_p \max(1 - \mu_{\tilde{p}}(p), \mu_{(-\infty,\tilde{t}]}(t + p)) = \sup_p \min(1 - \mu_{\tilde{p}}(p), \mu_{(-\infty,\tilde{t}]}(t + p)) \quad [11.30]$$

which brings the problem down to the example of maximization, relative to p and t, of:

$$\min(\mu_{[\tilde{r},+\infty)}(t), 1 - \mu_{(-\infty,\tilde{p}]}(p), \mu_{(-\infty,\tilde{t}]}(t + p)) \quad [11.30]$$

By considering linear membership functions, we thus find, by simple calculation of right segment intersection: t = 6 hours 52 minutes 30 seconds, which, as long as travel time does not last more than p = 1 hour 11 minutes 20 seconds, ensures a presence at the meeting no later than at 8:03:45, an acceptable delay, beyond which the decision-maker considers that he can justify this delay by transportation hazards. We thus obtain a good compromise, safe but realistic. Note that if we replace $1 - \mu_{(-\infty,\tilde{p}]}$ by $\mu_{\tilde{p}}$ in the problem reformulated by expression [11.30], we obtain the trivial optimistic solution $t = 7$, $x = 1$, which in practice is not plausible.

Calculating expression [11.30] makes it possible to handle uncertain execution times as a case of flexible execution times, as long as we replace the uncertainty distribution $\mu_{\tilde{p}}(p)$ by the (increasing) membership function of the complement of $(-\infty, \tilde{p}]$. We can, more generally, use the resolution tools and the approach described in section 11.4, even when tasks have ill-known execution times. However, we must then remember that in this case, task execution times thus obtained represent pessimistic duration previsions that we suppose to be sufficiently reasonable to occur in reality. The greater the global degree of satisfaction obtained, the more confidence we can place in these previsions, and the more robust against hazards are task starting times calculated by this approach.

This approach was extended to the job-shop problem under flexible constraints with ill-known execution times for tasks [DUB 95a] (see also [CHA 04]). Considering two tasks i and k on a single machine that can occur at the same time, a disjunctive constraint of non-overlapping between these tasks is confirmed with a degree of necessity equal to $\max(N(p_i \leq t_k - t_i), N(p_k \leq t_i - t_k))$. In addition, we have been able to extend some constraint analysis methods (see Chapter 5) to the fuzzy case in order to test, for each precedence conflict between tasks assigned to the same machine, whether due-date constraints enforce a precedence relation over another one, in the context of the maximization of the degree of satisfaction of the most violated constraint.

11.6. Conclusion: the potential contribution of possibility theory in scheduling

Possibility theory makes it possible to extend the constraint-based approach to scheduling problems so as to include the notion of preference. It also enables uncertainty to be represented for certain parameters such as processing times. As for the first aspect, the approach by flexible constraints has several specificities making it attractive in scheduling problems:

– it can easily express local preference specifications. These preferences come directly from the decision-maker or from the relaxation of constraints that are too tight. Flexible constraints are sometimes more expressive than global criteria that we find in literature. We can for example encode the problem of makespan minimization as a particular case of the flexible constraint problem;

– it assumes that violations of certain constraints are not compensated by total satisfaction of the others. We therefore do not minimize global cost, but we try to balance possible task due-dates. This comment clearly positions this approach in contrast with optimization of additive global criteria;

– it is totally in agreement with constraint-based analysis, of which this is a particular case. We can also directly use existing propagation tools, either by efficiently propagating local preferences, or by resolving the flexible constraint problem as a set of standard problems with hard constraints obtained by setting aspiration levels beforehand (which define hard constraints in the form of level cuts) [DUB 01a]. In fact, solutions to an optimal flexible scheduling problem are better described as solutions to a scheduling problem with hard constraints obtained by relaxing constraints as little as possible. The goal of a flexible constraint problem can therefore be seen as the specification of a standard problem with hard constraints making a trade-off between them in order to ensure the existence of a solution;

– although more elaborate than the approach by standard constraints, its complexity remains in the same range. It remains applicable to problems with similar size to those treated by the traditional constraint-based approach.

The notion of flexible constraint makes it possible to avoid two pitfalls of modeling by constraints: the case of the over-constrained problem where we sometimes only discover the inconsistency after many calculations; or the case of the under-constrained problem having too many solutions to be useful. Nevertheless, there are still very few studies using this approach. Although it seems promising, it still remains to be validated in practice, in terms of its effective power of expression as well as calculation times obtained, even if tests performed on 169 problems with 5 machines and 30 tasks in [FAR 97] seem conclusive. In particular, the use of flexible constraints can more quickly find an achievable solution to the problem (except for a very narrow zone of problems where consistency is very difficult to

prove). Other tests on larger examples with 10 machines have illustrated the same trend [FAR 97].

Possibility theory also offers a simple and gradual modeling of uncertainty in terms of task execution times. This modeling generalizes interval-based sensitivity analysis. It is different from the probabilistic approach based on hypotheses of independence and often assuming statistical knowledge. The possibilistic critical path method does not meet all difficulties often making a rigorous probabilistic approach unworkable. The latter approach, proposed a long time ago, must take into consideration the dependence between paths, contrary to the possibilistic approach. Even the determination of earliest starting time distributions is a difficult problem with a probabilistic approach, and we have seen that it is carried out simply in the context of fuzzy sets. Nevertheless, the determination of latest dates and floats in the possibilistic context cannot be carried out by traditional tools, and goes through the identification of parameter configurations accomplishing better and worse case situations. In the case where we have flexible constraints and uncertain knowledge at the same time, possibility theory proposes a moderately pessimistic criterion achieving robustness in the face of uncertainty, which constitutes an alternative to the expected utility criterion. From a formal point of view, this criterion is easily integrated in the context of flexible constraint satisfaction and propagation techniques.

11.7. Bibliography

[BEL 88] E. BENSANA, G. BEL, D. DUBOIS: "OPAL: a multi-knowledge based system for industrial job-shop scheduling", *Int. J. of Production Research*, vol. 26, no. 5, p. 795-819, 1988.

[BLA 86] BLAZEWICZ J., CELLARY W., SLOWINSKI R. and WEGLARZ J., "Scheduling under resource constraint – deterministic models", *Annals of Operations Research*, vol. 7, J.C. Baltzer AG, 1986.

[BOR 85] BORTOLAN G. and DEGANI R., "A review of some methods for comparing fuzzy subsets", *Fuzzy Sets and Systems*, vol. 15, p. 1-19, 1985.

[CHA 81] CHANAS S. and KAMBUROWSKI J., "The use of fuzzy variables in PERT", *Fuzzy Sets and Systems*, vol. 5, p. 11-19, 1981.

[CHA 98] CHANAS S. and KUCHTA D., "Discrete fuzzy optimization", in *Fuzzy Sets in Decision Analysis, Operations Research and Statistics*, R. Slowinski (ed.), p. 249-277, The Handbooks of Fuzzy Sets Series, Kluwer Acad. Publ., Boston, 1998.

[CHA 01] CHANAS S. and ZIELINSKI P. "Critical path analysis in the network with fuzzy activity times", *Fuzzy Sets and Systems*, vol. 122, p. 195-204, 2001.

[CHA 02a] CHANAS S. and ZIELINSKI P. "The computational complexity of the criticality problems in a network with interval activity times" *European Journal of Operational Research*, vol. 136, p. 541-550, 2002.

[CHA 02b] CHANAS S., DUBOIS D., ZIELINSKI P. "On the sure criticality of tasks in activity networks with imprecise durations*". IEEE Transactions on Systems, Man, and Cybernetics*, vol. 32, p. 393-407, 2002.

[CHA 03] CHANAS S., KASPERSKI A. "On two single machine scheduling problems with fuzzy processing times and fuzzy due dates" *European Journal of Operational Research*, vol. 147, p. 281-296, 2003.

[CHA 04] CHANAS S., KASPERSKI A. "Possible and necessary optimality of solutions in the single machine scheduling problem with fuzzy parameters" *Fuzzy Sets and Systems*, vol. 142, p. 359-371, 2004.

[CHE 92] CHEN S.-J. and HWANG C.-L., *Fuzzy Multiple Attribute Decision Making*, Lecture Notes in Economics and Mathematical Systems, vol. 375, Springer Verlag, Berlin, 1992.

[DUB 78] DUBOIS D. and PRADE H., "Algorithmes de plus courts chemins pour traiter des données floues", *RAIRO-Recherche Opérationnelle/Operations Research*, vol. 12, p. 212-227, 1978.

[DUB 88] DUBOIS D. and PRADE H., *Possibility Theory*, Plenum Press, New York, 1988.

[DUB 94] DUBOIS D. and KONING J.L., "A decision engine based on rational aggregation of heuristic knowledge"*, Decision Support Systems*, vol. 11, p. 337-361, 1994.

[DUB 95a] DUBOIS D., FARGIER H. and PRADE H., "Fuzzy constraints in job-shop scheduling", *Journal of Intelligent Manufacturing*, vol. 6, p. 215-234, 1995.

[DUB 95b] DUBOIS D. and PRADE H., "Possibility theory as a basis for qualitative decision theory", *14th IJCAI*, Montreal, Canada, p. 1924-1930, 1995.

[DUB 96] DUBOIS D., FARGIER H., PRADE H. "Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty", *Applied Intelligence*, vol. 6, p. 287-309, 1996.

[DUB 99] DUBOIS D. and FORTEMPS P., "Computing improved optimal solutions to fuzzy constraint satisfaction problems", *European Journal of Operational Research*, vol. 118, p. 95-126, 1999.

[DUB 00] DUBOIS D., KERRE E., MESIAR R. and PRADE H., "Fuzzy interval analysis", in *Fundamentals of Fuzzy sets*, D. Dubois, H. Prade (eds.), p. 483-581, The Handbook of Fuzzy Sets Series, Kluwer Acad. Publ., Boston, 2000.

[DUB 01a] DUBOIS D. and PRADE H., "Advances in the egalitarist approach to decision-making in a fuzzy environment", in *Dynamical Aspect in Fuzzy Decision Making*, Y. Yoshida, Ed., Physica-Verlag, Heidelberg Germany, Studies in Fuzziness and Soft Computing, vol. 73, p. 213-240, 2001.

[DUB 01b] DUBOIS D., PRADE H. and SABBADIN R., "Decision-theoretic foundations of qualitative possibility theory", *European Journal of Operational Research*, vol. 128, p. 459-478, 2001.

[DUB 03a] DUBOIS D., FARGIER H. and FORTEMPTS P., "Fuzzy scheduling: modeling flexible constraints vs. coping with incomplete knowledge", *European Journal of Operational Research*, vol. 147, p. 231-252, 2003.

[DUB 03b] DUBOIS D., FARGIER H. and GALVAGNON V., "On latest times and floats in activity networks with ill-known durations", *European Journal of Operational Research*, vol. 147, p. 266-280, 2003.

[DUB 05] DUBOIS D., FARGIER H., FORTIN J. Computational methods for determining the latest starting times and floats of tasks in interval-valued activity networks, *J. Intell. Manufacturing*, vol. 16, p. 407-422, 2005.

[ESQ 99] ESQUIROL P. and LOPEZ P., *L'ordonnancement*, Economica, Paris, 1999.

[FAR 97] FARGIER H., "Fuzzy scheduling: principles and experiments", in *Fuzzy Information Engineering: A Guided Tour of Applications*, D. Dubois, H. Prade and R.R. Yager (eds.), p. 655-669, Wiley, 1997.

[FAR 00] FARGIER H. and GALVAGNON V., D. DUBOIS "Fuzzy PERT in series-parallel graphs" *IEEE International Conference on Fuzzy Systems (*FUZZ-IEEE'2000), San Antonio (Texas), IEEE Press, p. 717-722, 2000.

[FOD 94] FODOR J. and ROUBENS M., *Fuzzy Preference Modelling and Multicriteria Decision Support*, Kluwer Acad. Publ., Dordrecht, 1994, section 7.2.2.

[FOR 96] FORTEMPS P. and ROUBENS M., "Ranking and defuzzification methods based on area compensation", *Fuzzy Sets and Systems*, vol. 82, p. 319-330, 1996.

[FOR 97a] FORTEMPS P. and HAPKE M., "On the disjunctive graph for project scheduling", *Foundations of Computing and Decision Sciences*, vol. 22, p. 195-209, 1997.

[FOR 97b] FORTEMPS P., "Job-shop scheduling with imprecise durations: a fuzzy approach", *IEEE Transactions on Fuzzy Systems*, vol. 5, p. 557-569, 1997.

[FOR 05] FORTIN J., ZIELINSKI P., DUBOIS D., FARGIER H., Interval analysis in scheduling, *Proc. of the 11th International Conference on Principles and Practice of Constraint Programming (CP'05),* Sitges, Spain, P. van Beek (eds.), Lecture Notes in Computer Science, Volume 3709, Springer, p. 226-240, 2005.

[FOR 06] FORTIN J., DUBOIS D., Solving Fuzzy PERT using gradual real numbers, *Proc. Starting AI Researcher's Symposium (STAIRS 2006),* Riva del Garda, Italy, L. Penserini, P. Peppas, A. Perini (eds.), IOS Press, Frontiers in Artificial Intelligence and Applications, p. 196-207, 2006.

[GRA 94] GRABOT B. and GENESTE L., "Dispatching rules in scheduling: a fuzzy approach", *International Journal of Production Research,* vol. 32, p. 903-915, 1994.

[HAP 94] HAPKE M., JASZKIEWICZ A. and SLOWINSKI R., "Fuzzy project scheduling system for software development", *Fuzzy Sets and Systems*, vol. 21, p. 101-117, 1994.

[HER 05] HERROELEN W., LEUS R. "Project scheduling under uncertainty: survey and research potentials" *European Journal of Operational Research*, vol. 165, p. 289-306, 2005.

[KAS 05] KASPERSKI A. "A possibilistic approach to sequencing problems with fuzzy parameters", *Fuzzy Sets and Systems*, vol. 150, p. 77-86, 2005.

[LOO 97] LOOTSMA F., *Fuzzy Logic for Planning and Decision-Making,* Kluwer Acad. Publ., Dordrecht, 1997.

[MAC 92] MACCAHON C.S. and LEE E.S., "Fuzzy job sequencing for a flow shop", *European Journal of Operational Research,* vol. 62, p. 294-301, 1992.

[MAC 93] MACCARTHY B. and LIU J., "Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling", *International Journal of Production Research*, vol. 31, p. 59-79, 1993.

[NAS 93] NASUTION S.H., "Fuzzy critical path method", *IEEE Transactions on Man, Machine and Cybernetics*, vol. 24, no. 1, p. 48-57, 1993.

[PIR 97] PIRLOT M. and VINCKE P., *Semiorders: Properties, Representations, Applications,* Kluwer Acad. Publ., Dordrecht, 1997.

[PRA 79] PRADE H., "Using fuzzy set theory in a scheduling problem: a case study", *Fuzzy Sets and Systems*, vol. 2, p. 153-165, 1979.

[ROM 90] ROMMELFANGER H.J., "Fulpal: an interactive method for solving (multiobjective) fuzzy linear programming problems", in *Stochastic versus Fuzzy Approaches to Multiobjective Mathematical Programming under Uncertainty,* R. Slowinski and J. Teghem (eds.), p. 321-330, Kluwer, 1990.

[ROM 94] ROMMELFANGER H.J., "Network analysis and information flow in fuzzy environment", *Fuzzy Sets and Systems*, vol. 67, p. 119-128, 1994.

[TUR 99] TURKSEN I.B. and FAZEL ZARANDI M.H., "Production planning and scheduling: fuzzy and crisp approaches", in *Practical Applications of Fuzzy Technologies*, H.-J. Zimmermann (ed.), p. 479-530, The Handbook of Fuzzy Sets Series, Kluwer Acad. Publ., Dordrecht, 1999.

[VAN 92] VAN LAARHOVEN P.J.M., AARTS E.H.L. and LENSTRA J.K., "Job-shop scheduling by simulated annealing", *Operations Research*, vol. 40, p. 112-125, 1992.

[WAN 01] WANG X., KERRE E. "Reasonable properties for the ordering of fuzzy quantities", *Fuzzy Sets and Systems*, 118, p. 375-405, 2001.

[WER 99] WERNERS B. and WEBER R., "Decision and planning in research and development", in *Practical Applications of Fuzzy Technologies*, H.-J. Zimmermann (ed.), p. 445-478, The Handbook of Fuzzy Sets Series, Kluwer Acad. Publ., Dordrecht, 1999.

[ZIE 05] ZIELINSKI P. "On computing the latest starting times and floats of activities in a network with imprecise durations" *Fuzzy Sets and Systems*, 150, p. 53-76, 2005.

Chapter 12

Real-Time Workshop Scheduling

12.1. Introduction

The real-time (or on-line) workshop scheduling problem consists of *permanently adapting* the execution terms of a set of jobs using a set of resources to the real situation of the system involved. This problem is different from the projected (or off-line) scheduling problem which is aimed at *projecting* the execution terms of a set of jobs using a set of resources over a finite or infinite horizon, based on hypotheses on the evolution of the system's situation over time. The chapter can be divided into two parts. The first part (sections 12.2, 12.3 and 12.4) focuses on positioning of real-time workshop scheduling in the production management system scheduling function. It shows the interest of the problem and, with a brief state of the art, the low number of studies addressing it in a methodological way. It also defines the decision variables, characteristics and objectives of the problem involved. The second part (sections 12.5, 12.6 and 12.7) proposes a real-time scheduling method in the context of workshops working on demand with small and medium-sized runs.

12.2. Interest and problem positioning

12.2.1. *The context of on demand production workshops*

For companies working on demand, the current context imposes production times that keep getting shorter, whereas the increasingly irregular demand makes it difficult to anticipate production. In addition, respect for announced manufacturing

---

Chapter written by Christian ARTIGUES and François ROUBELLAT.

delivery dates becomes essential in the supply chain context which controls manufacturer/supplier relations. In this type of environment, the workshop scheduling function plays a vital role. It is the one organizing fulfillment of the projected production schedule using workshop resources, integrating into this organization the unexpected or urgent orders as best as possible, and controlling the consequences of external or internal workshop uncertainties with increasingly tight production programs.

The scheduling problem for this type of company is characterized by a set of jobs (or manufacturing orders) to be produced over a given horizon issued from an MRP type plan [GIA 88]. Each job has a release date and a desired due date. The scheduling objective is to precisely allocate the operations which make up the jobs to workshop resources and to define their start dates. The consideration of real workshop characteristics imposes a satisfaction of complex constraints: multi-operation jobs with various sequencing constraints, operations requiring several resources for their completion, resource setup times or cost timetables associated with resources, etc. The necessity of having a dynamic vision of the scheduling function in order to react quickly to (internal or external) disruptions that the workshop may encounter gets added to these complex characteristics of the scheduling problem: unexpected jobs to perform, resource breakdowns, supply problems, lack of components, etc.

Note that the real-time scheduling problem is naturally encountered in computer systems (see, for example, [GOT 93, SIL 94, STA 95]). In fact, the dynamic arrival of jobs to be fulfilled (processed) and the need to react very quickly constitute vital characteristics of these systems. The scheduling problem involved is generally limited to a resource or several parallel resources (processors). In addition, the interruption of tasks (preemption) is often allowed, contrary to workshops. The set of tasks to be performed at a given moment is relatively low compared to what we encounter in production. In pipeline architectures, the cyclic character of tasks to be scheduled can sometimes simplify problem resolution (see [GOT 93] and Chapter 7).

12.2.2. *The different approaches to real-time workshop scheduling*

The problem involved here is a job shop scheduling problem which has been the subject of a very large number of studies, of which we can find a summary in [BLA 96] in its simplest form (see also Chapter 2). Roughly, in (job-shop) scheduling, we can distinguish two phases: a predictive or off-line phase which takes place before the production process actually starts, and a real-time or on-line phase which performs scheduling decisions in real time as the production process goes by. We cannot define the real-time scheduling approach independently of the selected off-line scheduling approach. We first describe two extreme cases that have

been widely studied in the literature and that illustrate well this relationship between off-line and real-time scheduling: the pure static case and the pure dynamic case [STA 95].

In the pure static case, a set of jobs is presumed known beforehand over a finite horizon. The traditional off-line approach consists of calculating a sequence of job-operations on each resource. In academic approaches, numerous exact or heuristic optimization algorithms are proposed, and some of them are described in Chapters 3, 4, 5 and 8 ([CAR 89, ADA 88, BAL 95, NOW 96, MAR 96, SCH 98, MAS 02]). In most production scheduling software packages, an approximate algorithm based on the use of one or more priority rules is used to solve the static problem (see [BLA 82] and Chapter 6). More recently, we find scheduling software packages based on more sophisticated approximate algorithms. A production software program based on the shifting-bottleneck heuristic [ADA 88] is presented in [MEE 96]. Another software package using the tabu search method (see [GLO 93, NOW 96] and Chapter 3) is described in [ART 97].

The real-time approach based on resolving the static problem consists of controlling the respect of proposed sequences and projected start times. New jobs included in the production plan or modifications of resource characteristics following disruptions are taken into consideration in a deferred way, from the moment when a new static problem is generated, based on a "rolling horizon" principle. A complete rescheduling is then carried out. One of the problems is to set the date from which a new schedule has to occur [OVA 94, VIE 03] and obviously this approach is well suited to the cases where the need for rescheduling is not too frequent. Another real-time approach is reactive scheduling where in the presence of an unexpected event the off-line schedule can be modified or repaired through local modifications.

In the pure dynamic case, the set of jobs is not known in advance and the scheduling horizon is infinite. When a job arrives, we must allocate its operations to resources and position them in relation to operations which have not yet started. Thus the on-line approach consists of following a certain *policy* for queue management for each resource. The off-line approach consists of selecting the best policy with the help of flow simulation of jobs in the workshop. The dynamic aspect of the problem can also involve resource availability, for which the possibility of breakdowns is then considered, following probabilistic models based on the mean time between failures or mean time to repair elements, for example [XIE 89]. This simulation takes into account dynamic aspects of the problem (disruptions, arrival of jobs, etc.) and makes it possible to compare the performances obtained for different policies in order to retain one. This is the one that will effectively be applied to build the workshop schedule in real-time, by queue management for each resource (see [BLA 82] and the simulation approach presented in Chapter 6). In addition

to simulation yielding mean and standard deviation performances of the selected policy, another role of the off-line phase in the pure dynamic case can be to evaluate the performance of the on-line policy in the worst-case through analytical techniques. In some particular cases, performance guarantees can be obtained [ALB 03].

The Kanban method, mainly designed for production lines [SHI 86], can also be presented as a dynamic real-time scheduling method decentralized by Kanban management terms of each manufacturing center. The limited number of Kanbans in a given manufacturing center notably avoids the uncontrolled increase of work-in-process.

Another approach for considering disturbances occurring during the execution of operations consists of considering operation processing times or other characteristics such as resource availability, as uncertain data during the solving of the off-line scheduling problem. The projected schedule, defined by considering uncertainties, has a certain robustness in relation to disruption. According to [HER 05], there are three categories of off-line scheduling approaches under uncertainty.

Fuzzy scheduling takes place within the context of the possibility theory; this approach is presented in [DUB 95] (see also Chapter 11).

Stochastic scheduling consists of minimizing the expected value of the objective function (usually the makespan) taking account of the processing time probability distribution [SKU 05]. As opposed to the pure dynamic case, the set of jobs, precedence constraints and resource requirement are assumed to be known in advance. As a typical work in this area, Stork presents in [STO 00] a branch and bound to calculate a pre-selective policy minimizing the expected makespan for the stochastic resource-constrained project scheduling problem.

A third approach to tackle uncertainty during the off-line scheduling phase deals with robust, or *proactive*, scheduling [DAN 95, WU 99, ART 03, ALO 05, AYT 05, HER 05]. This approach differs from the stochastic scheduling approach in the sense that the off-line schedule is generated with the objective of being relatively insensitive to the actual realization of the uncertain data, for example by minimizing the maximum distance between the realized schedule and the optimal one if all data were known *a priori*. The approach described in this chapter proposes an off-line scheduling method based on another concept related to robustness and is detailed in section 12.2.3.

When the off-line schedule has been generated with robustness considerations, it usually incorporates a certain degree of flexibility to self-adapt to unexpected events. In such cases, the repair procedure or reactive scheduling has to be carried

out only if the inherent flexibility is not able to absorb the disruptions. This is the approach selected in this chapter.

12.2.3. *An original approach*

In the late 1970s [DEM 77], a new approach gave birth to a category of proactive/reactive scheduling methods. The method was based on the observation that neither the standard static approach, nor the dynamic scheduling approach really satisfies real-time scheduling of job shops working on demand. The purely dynamic approaches are penalized by their ignorance of the manufacturing plan which, even though it will change during its execution, definitely exists in this problem. Global performance of resource queue management by priority rules or by a more sophisticated algorithm working with a smaller group of operations can turn out to be insufficient. The purely dynamic approach by definition allows very little anticipation for the use of resources. However, this anticipation can be vital in workshops because of the setup times needed on some machines, procurement management, etc. The static approaches can generate better solutions for holding delays, when they themselves are not solely based on priority rules. On the other hand, not considering the disruptions occurring in real-time can make the proposed solutions inapplicable and the additional optimization effort ends up being useless. In the case where a disruption leads to a complete rescheduling, the static approach also penalizes the anticipation because the proposed sequences can be totally different from one scheduling to another. This can go all the way to a loss of confidence in the scheduling system from its users. Both approaches also suffer from their directive aspect, leaving no freedom of action to deciders since they only propose one solution.

In parallel to the necessity of reactivity with regard to disruptions, anticipation and flexibility requirements for decisions, generally taken by people, make the real-time workshop scheduling problem much more complex than the real-time scheduling problem of a computer system. The problem with evaluating the performance of a real-time scheduling policy in a production system, and more generally of establishing a diagnostic following a bad result of this system, must also be taken into consideration. In this way, in a workshop, the decision makers have a tendency to attribute the fact that due dates are not respected to erroneous decisions or to a lack of personnel productivity instead of to the absence of an adapted real-time scheduling policy. The desire to meet due dates in this context often leads decision makers to release product manufacturing as soon as possible, which comes down to uselessly overloading the workshop, considerably increasing work in the process.

With the hypothesis that real-time scheduling is crucial for on-demand workshops, the approach presented in this chapter and initiated in [DEM 77] and [THO 80] attempts to take advantage of static and dynamic approaches discussed above and tries to avoid their main disadvantages. It is based on the above-presented online and off-line scheduling phases. The first consists of generating in the off-line scheduling phase not just one solution, but a set of eligible solutions from the initially known production plan. The notion of eligibility, which is the basis of this approach, stipulates that a solution is eligible if it respects all the problem's constraints, job delivery dates also being defined as constraints[1]. The second proposition consists of using this set of eligible solutions in real-time (on-line) in the context of an interactive decision support system for real-time scheduling. The objectives of this system are:

– to improve in real-time solutions proposed by the off-line phase by updating them according to decisions taken and significant events which occur (disruptions in particular) while measuring their eligibility at any moment;

– when workshop status requires that a decision be taken, to help in the choice of this decision among the possible choices at that moment, by relying on updated solutions;

– in case of loss of eligibility following disruptions, to propose local modifications on the solution set to retrieve eligibility. The local character of modifications guarantees speed of execution and low disruption (robustness) from all proposed solutions. In this way, there is no need to reschedule as long as the set of solutions can remain eligible. Rescheduling only becomes necessary when the system can no longer find eligibility with the help of local modifications, which settles the problem of determining a rescheduling period which occurs when rolling horizon procedures are considered.

Note that since the early works of [DEM 77] and [THO 80], the method has been studied in several contexts and various forms [THO 88, ERS 89, LEG 92, BIL 96a, BIL 96b, ART 97, ART 99, WU 99, ART 03, ALO 05, BRI 07] and that a real-time production scheduling software has been issued from this work [THO 88] (see also section 12.8). More precisely, the method presented in this chapter is based on [ART 97].

12.3. Modeling and dynamic of scheduling problem considered

We can define a workshop as a system made up of basic entities which are operations and resources. We can associate characteristic parameters with each

1 The notion of eligibility concerning constraint propagation in scheduling is described in more detail in Chapter 5, section 5.3.

entity. Scheduling problem solving consists of defining the relational terms of these entities in order to complete a series of given jobs. Due to the dynamic approach chosen, the association of a variable *state* over time for each entity is vital to follow up the global workshop state and real-time decision-making. An entity goes from one state to another following an *event* or a *decision* the occurrence of which is controlled by *constraints*. We observe three fundamental entities in the production system: resources, production operations, setup operations, described in sections 12.3.1, 12.3.2 and 12.3.3 respectively.

12.3.1. *Resources*

Resources constitute workshop production components and are its fundamental element. The issue with the scheduling problem comes from the fact that their capacity is limited. In this chapter, we only consider renewable resources, i.e. where availability is completely returned after each operation. We distinguish *disjunctive resources*, which can only perform one operation at a given moment, from *cumulative resources* performing a limited number of operations at a given moment. We consider that a cumulative resource R_k is made up of a number of resource units $A_k > 1$ called capacity; each resource unit can be assimilated to a disjunctive resource. By extension, any disjunctive resource R_k has a unit capacity $A_k = 1$. In approaches generally encountered for the job shop problem, disjunctive resources are the only ones considered (see Chapter 2), cumulative resources being only considered for the resource-constrained project scheduling problems (see [HER 98] and Chapter 9). In reality, we often encounter both types of resources in shops: machines and tools can be represented by disjunctive resources, whereas operator teams can be represented by cumulative resources.

The maximum capacity of a resource may not always be available. Machines may have maintenance periods and operators do not work 24 hours a day. Most resources follow a *timetable* and at any instant t, capacity $A_k(t)$ of a resource satisfies $A_k(t) \leq A_k$.

At a given time period t, the state of the "resource unit" entity of a resource R_k is defined by the pair $\{E_1(k, t), E_2(k, t)\}$. Component $E_1(k, t) \in \{\text{LI, OC}\}$ defines the state of the resource unit in relation to operation fulfillment:

– *idle* state (LI) indicates that the resource unit is not occupied with the execution of an operation;

– *occupied* state (OC) indicates that the resource unit is busy with the execution of an operation.

Component $E_2(k, t) \in \{\text{PN, NP}\}$ is a disruption indicator encountered by the resource unit, called *breakdown*:

– *breakdown* state (PN) indicates that the resource unit no longer works, following a hazard linked to the resource;

– *non-breakdown* state (NP) indicates that the resource unit works properly.

Two additional state components will be introduced to take into consideration setup activities for a disjunctive resource (see section 12.3.3).

12.3.2. *Production operations*

Jobs to be fulfilled are made up of operations, each job i having an availability release date r_i and a delivery date d_i. $o(i, j)$ represents operation j of job i with $1 \leq j \leq n_i$. Precedence relations exist between operations of one job and do not necessarily constitute a total order between these operations, which is different from traditional job shop models. In fact, precedence relations between the different operations of job i can be represented by an operation-on-node graph G_i. In this way, for an operation $o(i,j)$, the set $pred(i,j)$ represents all operations immediately preceding $o(i, j)$ within job i. $succ(i, j)$ represents all operations that are direct successors of $o(i,j)$ in job i. These generalized precedence relations enable modeling of assembly problems [SCH 98, DAU 98] and in a more general way, non-linear routings [ART 97].

For their execution, production operations use one or more resources simultaneously (multi-resource context). For example, a production operation can simultaneously require the use of a machine, a tool or two operators.

In addition, there may be several terms of use for resources for a single production operation, particularly when a choice of different resources is possible for executing a production operation. Each resource use term is called a mode (denoted as m). An operation (i, j) may have $m_{ij} \geq 1$ mode(s), each mode m $(1 \leq m \leq m_{ij})$ defining:

– a number of resource units $a_{ijkm} \geq 0$ used on each resource R_k in the workshop;

– a duration p_{ijm}.

For example, a turning operation can be executed by numerically controlled lathe A (mode 1: one resource) *or* by numerically controlled lathe B (mode 2: one resource) *or* by an operator using a manually controlled lathe (mode 3: two resources).

With reference to models encountered in literature, multi-resource and multi-mode characteristics define the workshop problem involved here as a multi-project scheduling problem with limited resources and multiple modes [HER 98].

At a given time t, the state of a production operation is defined by the pair $\{O_1(i, j, t), O_2(i, j, t)\}$. Component $O_1(i, j, t) \in \{\text{BL, AP, DI, EC, TR}\}$ represents the operation situation in the job completion, i.e. its state of progress:

– *available* state (DI) indicates that execution of the production operation can start, parts are waiting in front of the resource(s);

– *in process* state (EC) indicates that execution of the production operation is in process;

– *finished* state (TR) indicates that execution of the production operation is finished for the job involved;

– *approaching* state (AP) is only defined for production operations with at least one preceding operation and indicates that any preceding operation is either finished, or "in process", but that there is at least one in the *in process* state;

– *blank* state (BL), also defined for operations with at least one previous operation, indicates that at least one of these previous operations is not finished or in process.

The second component $O_2(i, j, t) \in \{\text{BQ, NB}\}$ is an indicator of a disruption that the operation encountered, called "blocking", which prohibits any possibility of operation execution, regardless of the state of resources necessary for its execution (which may correspond to a lack of components for example). It is a disruption linked to the operation and not to resources that it requires. Two states are possible:

– *blocked* state (BQ) indicates that the production operation cannot be executed because of a blocking;

– *non-blocked* state (NB) indicates that the production operation is not subject to the blocking hazard.

12.3.3. *Setup operations*

Before the execution of a production operation, certain resources require the completion of a setup activity with the goal of getting them into a configuration compatible with the production operation to be fulfilled, called *setup state*. This setup activity may consist of one or more cleanings, adjustments, equipment assembly (tools, tooling) specific to the production operation to be executed, disassembly of equipment used by the production operation finished, etc. Depending on the context, this setup activity may be integrated in the value given for the

duration of an operation, or be explicitly taken into account. The first context is valid if setup time is low compared to production time, if setup requires the presence of parts to be completed, if it requires no resources except for the ones used for production (same machine *and* same operators for example) and if setup time does not depend on sequence of operations executed, or if setup can be carried out in concurrent operation time, i.e. without occupying resources. The objective here is to take into consideration a second context for which the setup activity can be anticipated before the arrival of parts for the production operation, and does not necessarily involve the same set of resources, but occupies this set of resources for a period time depending on its initial setup state and the setup state to be reached.

Traditional solving approaches (for a state of the art, please refer to [ART 97]) consider setup activity by introducing setup costs linked to changes in production type, or setup times occupying resources between two different types of production. Several types of setup times are found in literature. In general, the setup time considered immobilizes a single disjunctive resource, in the context of one-resource [GAV 65], parallel resource [GUI 91, MON 93], flow shop [PRO 91] and less often job shop [OVA 94] problems with assembly constraints (see [SCH 98]), or project scheduling (see [KOL 95], p. 177). We observe:

– resource assembly time independent of the sequence which immobilizes a resource before execution of an operation and only depends on this operation and the resource involved;

– resource disassembly time independent of the sequence immobilizing a resource after execution of an operation and only depends on this operation and the resource involved;

– resource setup time depending on the sequence which immobilizes a resource between the execution of two operations and depends on the resource involved, previous and next operations. In certain cases, each production operation is associated with a family and setup times are defined in relation to families.

The model retained in this chapter to describe the setup activity refines these traditional models, in order to take into consideration more precisely the multi-resource characteristics of this activity generally encountered in reality. With regards to the notion of setup, production operations are distributed into families (set F) and workshop resources can be divided into two subsets:

– set Rp of *main* resources contains all resources which may require a setup activity prior to execution of a production operation, i.e. for which the notion of setup state has a meaning; for simplification purposes, we consider that a main resource must be disjunctive;

– set *Rc* of *complementary* resources is the set of resources for which the notion of setup state makes no sense, i.e. they never require setup. They can be disjunctive or cumulative.

Each production operation belonging to a certain family and needing a set of main resources, the necessary setup activity to bring all required main resources in the configuration enabling the execution of this operation, can be broken down into three basic *setup operations*:

– family change operation, denoted as $s(f, f', u)$;

– multi-resource assembly and disassembly operations respectively noted as $s(-, u)$ and $s(u, -)$.

The family change operation $s(f, f', u)$ makes it possible to simultaneously move a set of main resources $u = \{R_{k1}, \ldots, R_{kr}\}$, $r \geq 1$, from one setup state compatible with family $f \in F$ to a setup state compatible with family $f' \in F, f' \neq f$. According to the same principle as with a production operation, several modes $m_{s(f, f', u)}$ of execution of a family change operation can exist. Each mode involves a different set of *complementary resource* units required by the family change operation. Operation duration depends on the mode chosen $m \in \{1, \ldots, m_{s(f, f', u)}\}$, of family f, of family f' and is noted as $p_{s(f, f', u)m}$. The family change operation occupies all main resources of set u and a number of resource units for each complementary resource R_k equal to $a_{s(f, f', u)km} \geq 0$.

Resource assembly operation $s(-, u)$ switches a set of main resources $u = \{R_{k1}, \ldots, R_{kr}\}$, $r > 1$, from "isolated" state to "associated within set u" state. There may be several modes $m_{s(-, u)}$ of execution of the assembly operation. Each mode involves a different set of complementary resources required by the assembly operation. Operation duration depends on the mode chosen $m \in \{1, \ldots, m_{s(-, u)}\}$, and is denoted as $p_{s(-, u)m}$. The assembly operation occupies all main resources of set u and a number of resource units on each complementary resource k given by $a_{s(-,u)km} \geq 0$.

Finally, resource disassembly operation $s(u, -)$ switches a set of main resources $u = \{R_{k1}, \ldots, R_{kr}\}$, $r > 1$, from "associated within set u" state to "isolated" state. There may be several modes $m_{s(u,-)}$ of execution of the disassembly operation. Each mode involves a different set of complementary resources required by the disassembly operation. Operation duration depends on the mode chosen $m \in \{1, \ldots, m_{s(u, -)}\}$, and is denoted as $p_{s(u, -)m}$. The disassembly operation occupies all main resources of set u and a number of resource units on each complementary resource k given by $a_{s(u, -)km} \geq 0$.

This original model introduces the setup operation notion, thus generalizing the traditional notion of setup time, in order to take into account the multi-resource context. The definition of these operations for setup is greatly justified for the consideration of complementary resources which can be different for setup and for production. The sequence of setup operations necessary prior to a production operation varies according to the initial setup state for each main resource used.

The setup state of a main resource k at a given time t is defined by the pair $\{S_1(k, t), S_2(k, t)\}$. Component $S_1(k, t) \in \{f \mid f \in F\} \cup \{(f, f') \mid f, f' \in F\}$ traditionally defines the setup state of R_k in relation to set F of production operation families:

– $S_1(k, t) = f$ indicates that resource k is in a state compatible with any production operation of family f;

– $S_1(k, t) = (f, f')$ indicates that resource k is moving at time t from a state compatible with family f to a state compatible with family f'.

$S_2(k, t)$ defines the state of *association* of R_k with a set of other main resources; this state is directly linked to the multi-resource context retained for production and setup operations. We observe the following cases:

– $S_2(k, t) = \varnothing$ indicates that resource k is not associated with any other main resource; it is in the *isolated* state;

– $S_2(k, t) = u$ indicates that resource k is *associated* within the set of main resources $u = \{R_{k1}, \ldots, R_{kr}\} \subset Rp$, with $k \in u$; it is therefore ready to execute any production operation simultaneously requiring all resources of u;

– $S_2(k, t) = (–, u)$ indicates that the resource is switching from "isolated" state to "associated within set $u = \{k_1, \ldots, k_r\}$" state;

– $S_2(k, t) = (u, –)$ indicates that the resource is switching from "associated within set $u = \{k_1, \ldots, k_r\}$" state to "isolated" state.

The generation of setup operations is dynamic and depends on sequencing and allocation decisions presented in the following section. This dynamic character makes this setup operation a generic operation with a "physical" existence only when it is in progress, contrary to production operation with an approaching, availability state, etc. For each generic setup operation (change of family, assembly, disassembly) s, we associate state $S_3(s, t) \in \{BQ, NB\}$ indicating if the operation is blocked or not (because of lack of a component to complete this setup, for example).

12.4. Decisions, events and constraints

This section contains the description of decisions and events, and their impact on the evolution state of entities presented in the previous section and constraints that they have to face.

We consider two types of decision significant for real-time scheduling concerning production operations:

– the *start* decision $E(o, m)$, consisting of allocating to operation o resources defined by the m mode and to start its execution, switching o from state DI (available) to state EC (in process) and involved resource units from state LI (idle) to state OC (occupied);

– the *interruption* decision $I(o, m)$ consisting of suspending the execution of an operation with mode m; this decision frees resource units involved in this execution.

The start decision $E(o, m)$ can only happen if each resource unit involved is idle (LI) and not in breakdown (NP), if the operation is available (DI) and non-blocked (NB), if the set of main resources used is associated and if each main resource is prepared for the operation family. On the other hand, no specific constraint influences the interruption decision.

We consider two types of decision significant for real-time scheduling concerning setup operations:

– the *start* decision $E(s, m)$, consisting of starting execution of setup operation s in mode m, switching resource units of each main and complementary resource involved from state LI (idle) to state OC (occupied) and each main setup resource from initial setup state to in process setup[2];

– the *cancellation* decision $A(s, m)$ which interrupts the execution of a setup s operation; this decision frees resource units (main and complementary resources used) involved and brings main resources back to the initial setup state.

The start decision *E(s, m)* can only happen if each resource unit involved is idle (LI) and not in breakdown (NP), if the setup operation is not blocked and if the set of main resources used is in the initial required setup state. For the cancellation decision, no constraint is defined by simplification.

Significant events for real-time scheduling can be divided into two classes: unexpected events and expected events. We consider two types of unexpected events that we can also call disruptions:

---

2 The “initial” and “in process setup” states vary according to the setup operation involved: change of family, assembly or disassembly (see section 12.3.3).

– the *breakdown* event which makes the use of a certain number of resource units a of a resource R_k temporarily impossible for any production or setup operation; this event is denoted as *P(k, a)*;

– the *blocking* event which prevents the execution of a production o (or setup s) operation independently of the resources that it uses (lack of procurement, absence of a non-modeled resource, etc.); this event is denoted as *B(o)* (respectively *B(s)*).

There are three types of expected event, each is the consequence of an unexpected event or decision:

– the *end of execution* event for production o (or setup s) operation is expected after the type *E(o, m)* (respectively *E(s, m)*) decision, actually at a date equal to the date of this decision plus operation duration p_o (p_s); this event frees resource units involved for a production operation and in the case of a setup operation, switches main resources to the final setup state associated with this operation;

– the *end of breakdown* event of a number a of resource units for resource k is expected after occurrence of a type *P(k, a)* event with or without an estimated breakdown time; it is denoted as *FP(k, a)*;

– the *end of blocking* event of production o or setup s operation is expected after occurrence of a type *B*(o) (respectively *B(s))* event with or without an estimated blocking time; it is denoted as *FB(o) (FB(s)*).

Any execution end event can only happen if each resource unit occupied is not in breakdown (NP) and if the in process operation is not blocked (NB).

The state sequence of the different entities according to events and decisions, as well as constraints weighing on these events and decisions characterizing the scheduling problem, can be modeled by a Petri net[3]. A Petri net is a network containing two types of nodes. Events and decisions are represented by “transition” nodes and states of entities by “place” nodes. We can find a detailed representation of such a network in [ART 97].

12.5. Models for off-line and on-line scheduling

By taking into consideration constraints presented in section 12.4, and knowing the current state of the workshop defined by the set of states of different entities presented in the previous section, the scheduling objective is to reach a final eligible state. This state then corresponds to transferring to the “finished” state the set of production operations before job delivery dates.

3 We can find a presentation on Petri nets in Chapter 7.

The bias of the method proposed is to consider, for projected scheduling, a deterministic model in which unexpected events are not considered, neither is the decision to interrupt an operation, since preempting an operation is not authorized in this scheduling phase. The goal of off-line scheduling is to plan a set of start decisions leading to a final eligible state, while remaining as flexible as possible. The real-time scheduling objective is to help in choosing the best decision at any time because of the state of the workshop and off-line schedule. In the case of disruption, real-time scheduling reacts by using available flexibility and if this disruption compromises eligibility of all expected decisions, by possibly proposing decisions of production operation interruption or cancellation of setup operations. These propositions then have the goal of locally modifying the set of decisions expected by projected scheduling in order to recover eligibility.

In order to bring out flexibility for its use with real-time scheduling during off-line scheduling, the approach proposed in section 12.2.3 is intended to define a set of solutions instead of just one as with traditional approaches. This inventive approach is now presented, in the context of the scheduling problem involved.

12.5.1. *Groups of interchangeable operations*

In order to facilitate real-time use of the set of schedules obtained from the projected scheduling procedure, emphasis on sequential degrees of freedom for the execution of production operations, easy to understand and use in real time, was preferred. This has led to introducing the concept of "sequence of interchangeable operation groups", in order to retain sequential indeterminism for production operations belonging to the same group [DEM 77]. A total order relation exists between any two operations belonging to two consecutive groups in the sequence. In terms of constraint propagation for other operations (see Chapter 5), the least favorable sequence of production operations, i.e. the most restricting start decisions involving other operations, is always considered. In this way, we ensure that start times associated with operations, i.e. projected dates for start decisions, will be compatible with any permutation selected in each group. In order to facilitate understanding of the set of solutions thus proposed, the scheduling method is only authorized to interchange production operations allocated to the same resources and for each resource, to the same resource units. In addition, it can only include in a group production operations from the same setup class. Finally, setup operations to be executed cannot be included in a group of interchangeable operations.

12.5.2. *Operation-on-node graphs*

The definition of interchangeable operation group sequences offers degrees of freedom of a sequential nature for operation start decisions. Temporal degrees of freedom, preserving sequential degrees of freedom, can also be highlighted by characterizing the start date of a production or setup operation[4] o with the help of an earliest start date r_o and a latest end date d_o. According to one of the two dates involved, a sequence of groups, i.e. a set of scheduling problem solutions, can be represented by two operation-on-node graphs for calculating these dates by potential propagation [ROY 70].

For earliest start date calculation, graph $G = (X, U)$ is defined in the following way. The set of nodes X is such that each projected production operation is a node and each necessary setup operation is also a node. For each job i, we add a fictitious operation $o(i, n_{i+1})$ following any operation of this job without successor in the routing. We also add to the graph original time node O and a node H to measure eligibility. The set of arcs U represents the set of precedence constraints . We define three types of arcs; the first two are only defined for production operations:

– constraints linked to the routing: a "routing" type arc is defined between two nodes corresponding to two consecutive production operations in the job routing; this arc is valued by duration of operation corresponding to the original node; a routing type arc is defined between node O and each node corresponding to the first operation in a job i, valued by r_i; a routing type arc is defined between each node corresponding to the last operation $o(i, n_i + 1)$ of a job i and node H, valued by $-d_i$;

– constraints linked to routing and to considering the most unfavorable permutations within one group ("group" type arcs): a group type arc is defined between two nodes if the destination node corresponds to a production operation where a previous operation in the routing belongs to the same group as the operation corresponding to the original node. This arc is valued by the sum of durations of production operations belonging to the same group as the production operation corresponding to the original node, i.e. duration of this group;

– constraints linked to resource sharing ("resource" type arcs): a resource type arc is defined between two nodes x_1 and x_2 in the following four cases: either x_1 and x_2 correspond to two production operations whose groups are (directly) consecutive on a set of non-empty resource units; or x_1 and x_2 correspond to two setup operations directly consecutive on a set of non-empty resource units; or x_1 corresponds to a setup operation, x_2 corresponds to a production operation and group x_2 is directly consecutive to the setup operation x_1 on a set of non-empty resource units; or x_1 corresponds to a production operation, x_2 corresponds to a setup

4 Subsequently, o represents an operation without specifying if it is a setup or production operation, whereas $o(i, j)$ exclusively represents a production operation.

operation and this operation is directly consecutive to group x_1 on a set of non-empty resource units. In the case where x_1 corresponds to a production operation, the arc is valued by duration of its group (which is the sum of the durations of the operations inside the group); in the case where x_1 corresponds to a setup operation, the arc is valued by its duration. Earliest start dates for all operations can be obtained by potential propagation in this graph from node *O* with which we associated zero potential. Potential obtained for node *H* is the maximum lateness denoted as $L_{\max}$. It gives information on eligibility of the group sequence.

Contrary to the traditional method [ROY 70], we cannot calculate the latest end date by reverse propagation in graph *G*, because group type arcs do not correspond to the expression of the most unfavorable permutation constraint for end dates. We thus have to use another graph containing (in reverse) the exact same routing and resource type arcs but different group type arcs [THO 80].

However, when the number of operations in a group is large, the number of group type arcs can be very significant. Fortunately, it is not necessary to represent group type arcs, the knowledge of operation membership to a group and routing type arcs are sufficient for representing group type arcs implicitly. We can thus blend both graphs into a single one called a "group graph". This graph which prevents combinatorial explosion of group type arcs is presented with the help of this next example. The algorithm enabling the calculation of earliest start and latest end dates from this graph is presented in section 12.5.3.

EXAMPLE.– A workshop is made up of 9 resources, 7 of which are disjunctive, representing two presses (denoted as *PR*1 and *PR*2) , two molds (denoted as *MO*1 and *MO*2), three operators (denoted as *O*1, *O*2 and *O*3) and two cumulative resources representing two operator teams (*EQ*1 and *EQ*2), where each team contains two operators (two resource units, notation *ldc* in the diagram). We consider five jobs with an earliest start date equal to 0 and with linear routing[5]. In this example, all production and setup operations only have one mode of execution. Job characteristics are presented in Table 12.1.

i	d_i	j	F_{ij}	Resources	p_{ij}	j	F_{ij}	Resources	p_{ij}
1	20	1	A	*PR*1, *MO*1, *O*1	2	2	B	*PR*1, *MO*2, *O*3	1
2	20	1	C	*PR*2, *MO*2, *O*2	1	2	B	*PR*1, *MO*2, *O*3	2
3	5	1	C	*PR*2, *MO*2, *O*2	2	-	-	-	-
4	10	1	-	*EQ*1 (2 *ldc*)	1	2	-	*EQ*2 (1 *ldc*)	4
5	10	1	-	*EQ*1 (2 *ldc*)	2	2	-	*EQ*2 (1 *ldc*)	6

Table 12.1. *Job characteristics*

5 G_i is a chain.

Association u	Resource Disassembly $s(u, -)$		Resource Assembly $s(-, u)$	
	Duration	Complementary	Duration	Complementary
{*PR*1, *MO*1}	5	*EQ*1 (1 *ldc*)	4	*EQ*1 (2 *ldc*)
{*PR*1, *MO*2}	8	*EQ*1 (1 *ldc*)	2	*EQ*1 (2 *ldc*)
{*PR*2, *MO*1}	2	*EQ*1 (1 *ldc*)	6	*EQ*1 (2 *ldc*)
{*PR*2, *MO*2}	4	*EQ*1 (1 *ldc*)	8	*EQ*1 (2 *ldc*)

Table 12.2. *Resource assembly and disassembly operation characteristics*

Change	Duration	Complementary	Change	Duration	Complementary
s(*A, B, u*)	3	*EQ*2 (1 *ldc*)	*s*(*C, A, u*)	5	*EQ*2 (1 *ldc*)
s(*A, C, u*)	4	*EQ*2 (1 *ldc*)	*s*(*C, B, u*)	6	*EQ*2 (1 *ldc*)
s(*B, A, u*)	1	*EQ*2 (1 *ldc*)			
s(*B, C, u*)	3	*EQ*2 (1 *ldc*)			

Table 12.3. *Family change operation characteristics*

For example, job 1 has a delivery date equal to 20 and two operations: $o(1, 1)$ duration equal to 2, of family *A*, allocated to resources *PR*1, *MO*1, *O*1 and operation $o(1, 2)$ of family *B*, duration equal to 1 and allocated to resources *PR*1, *MO*2, *O*3. Concerning setup activities, press and mold resources (*PR*1, *PR*2, *MO*1 and *MO*2) are the main workshop resources liable to have setup activities. Operator resources (*O*1, *O*2, *O*3, *EQ*1 and *EQ*2) are complementary resources. The distribution of production operations into families is indicated in Table 12.1. We can observe that manufacturing order operations 4 and 5 have no family. In fact, they are only allocated to complementary resources. Resource assembly and disassembly operation characteristics are presented in Table 12.2. For example, *PR*1 resource assembly operation with *MO*1 lasts for 4 units of time and uses 2 resource units from complementary resource *EQ*1. The characteristics of family change operations are given in Table 12.3, where *u* can be replaced by any resource in the set {*PR*1, *PR*2, *MO*1, *MO*2} or any association of a set element {*PR*1, *PR*2} with any element of the set {*MO*1, *MO*2}. In this way, for isolated resource *PR*1, changing from family *A* to family *B* lasts for 3 units of time and uses 1 resource unit of complementary resource *EQ*2.

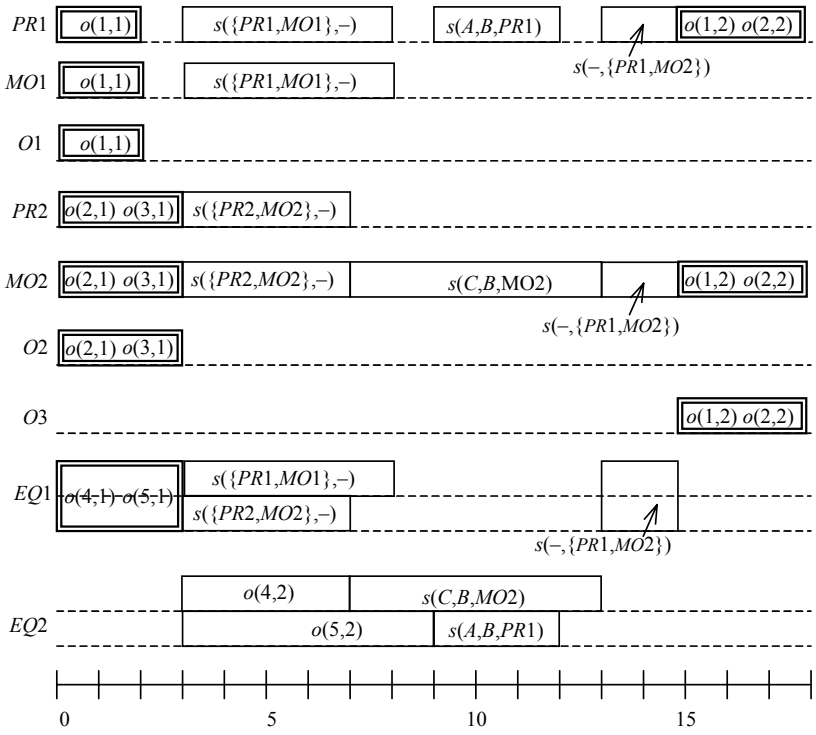


Figure 12.1. *Example of an eligible schedule set*

The sequence of groups presented in Figure 12.1 in the form of a Gantt chart represents a set of solutions to the problem thus defined. The sequence of groups contains 3 groups with more than one operation: {o(2, 1), o(3, 1)}, {o(1, 2), o(2, 2)} and {o(4, 1), o(5, 1)}. Therefore, 8 schedules are represented.

Necessary setup operations are represented by simple line rectangles, whereas interchangeable production operation groups are represented by double line rectangles. In this Gantt chart, operations and groups are scheduled the earliest. We thus observe that the sequence of proposed groups is eligible by looking at job due dates. The two graphs used for calculating earliest start and latest end dates for operations are represented implicitly by the single graph in Figure 12.2.

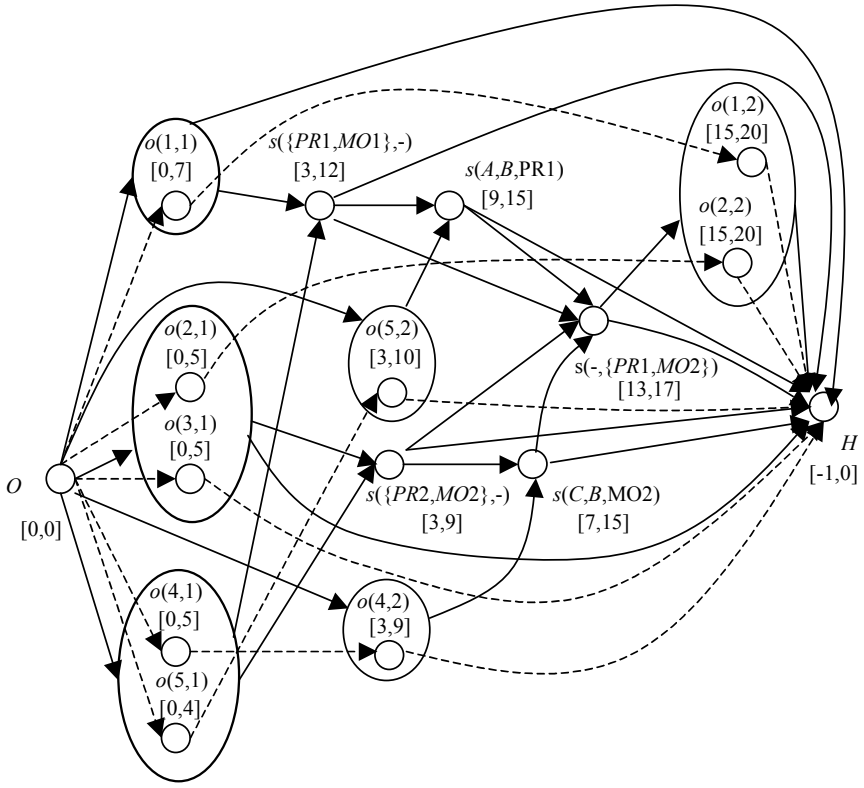


Figure 12.2. *Graph representing sequence of groups in Figure 12.1*

In this graph, we have two types of nodes: "operation" and "group" nodes. We only represent resource type arcs (bold line) and routing type arcs (dotted line) and we include nodes representing production operations located in one group within the node representing the group. We define routing type arcs between two "operation" nodes and resource type arcs between two "group" nodes. This representation makes it possible to reduce the number of arcs and to highlight the group concept in the operation-on-node graph. For example, the group type arc existing between node $o(3, 1)$ and node $o(2, 2)$ for earliest start date calculation is implicitly represented by the routing type arc between $o(2, 1)$ and $o(2, 2)$ which traverses the node representing group $\{o(2, 1), o(3, 1)\}$. For calculating latest end dates, a "group" type arc exists between node $o(2, 2)$ and node $o(1, 1)$, but it is implicitly represented by the routing type arc between $o(1, 1)$ and $o(1, 2)$, which "passes through" the node representing group $\{o(1, 2),o(2, 2)\}$. The method for calculating dates associated with operations and verifying eligibility of a sequence of groups is presented in the next section.

12.5.3. *Generic graph methods*

Potential propagation method

From a sequence of groups given by a graph, we use a potential propagation method to calculate the earliest start and latest end dates of the different operations, represented in square brackets in Figure 12.2. For earliest start date calculation, we allocate zero potential to the original O node and we propagate this potential with an extension of the Bellman algorithm (see Chapter 2) with a recursive formulation as follows: the potential of (production or setup) "operation" node o of the graph, i.e. the earliest operation start date r_o, is the largest value between:

– the earliest end date of any operation $o(i, j)$ whose node is the origin of a routing type arc having node o *as destination*, in the most unfavorable permutation situation in group g_{ij} containing $o(i, j)$:

$$r_o \geq r_{ij} + p_{ij}$$

$$r_o \geq \max\{r_{xy} \mid o(x, y) \in g_{ij},\ o(x, y) \neq o(i, j)\} + \sum\{p_{xy} \mid o(x, y) \in g_{ij}\}$$

– the largest earliest end date of operations included in any group g whose node is linked to the group of node o by a resource type arc:

$$r_o \geq \max\{r_{o'} \mid o' \in g\} + \sum\{p_{o'} \mid o' \in g\}$$

In the presence of timetables, we must add to these values durations of inactive periods of resources used by the operation or the group during the period involved.

In the graph in Figure 12.2, we calculate in this way a maximum lateness of –1 (advance), since job 5 ends on date 9. Latest end dates are obtained in a symmetric way by starting the propagation from node H, by allocating for example to node H a latest end date equal to $\max(0, L_{\max})$, as was done in Figure 12.2. For these calculations, the graph model requires no distinction between setup and production operations.

Insertion method for a production operation

The model based on graphs makes it possible to analyze the impact of a local modification of the series of solutions on eligibility, notably when this modification is caused by an insertion. We can define the insertion problem of a production operation in a set of schedules represented by a sequence of groups in the following way. Given a scheduling problem A defined by a set of resources and jobs, a sequence of groups S_A eligible for problem A, a production operation $o(i, j)$ not in A and with an earliest start and latest end date, the insertion problem of $o(i, j)$ comes down to proposing a new eligible sequence of groups S_B, or the closest possible to eligibility for problem B defined by adding $o(i, j)$ to A. To solve this problem, a purely dynamic approach consists of solving problem B with no consideration for

S_A. In contrast, the method proposed here uses this sequence as insertion support for operation $o(i, j)$. We set an objective of inserting $o(i, j)$ in the set of schedules:

– by respecting sequence S_A, i.e. without changing the relative order of operations already present, or their allocation;

– by minimizing violation of latest end dates of $o(i, j)$ and already sequenced operations in S_A, $o(i, j)$ satisfying its earliest start date.

Preserving S_A has the double purpose of not provoking major changes in the sequence, which may not be accepted well in real time in a workshop, and making problem *B* much simpler to solve because of the significant reduction of all possible solutions.

Details of this method will not be described here. For an idea of its principle, we use each resource type arc u from graph G modeling the current set of solutions S_A as a characteristic element of a position of insertion. In fact, there is a positive or zero time interval between the earliest end date of group $g1$ corresponding to the origin node of u and the latest start date of group $g2$ corresponding to destination node of u. In addition to this interval, called the *time window*, arc u also defines an *resource occupation*, i.e. the resource unit subset that group $g2$ directly recovers after being used by $g1$. These two elements, characterizing the *insertion interval*, totally define maximum allocation of resource units to an operation if we settle for inserting it between $g1$ and $g2$, and its maximum duration preserving eligibility. Generally, it is necessary to jointly use several resource type arcs to give an operation all the resource units that it requires. Provided these arcs are compatible, i.e. not located on the same route in graph G, the insertion interval represented by these arcs has a resource occupation equal to the union of the resource occupations of its arcs and a time window equal to the intersection of windows associated with its arcs. The main concern with the insertion problem is to avoid listing all possible insertion intervals and to verify compatibility of arcs making up each insertion interval. In the case where the operation to be inserted cannot generate a setup activity, a polynomial algorithm has been defined to find the optimal insertion interval, i.e. minimizing maximum violation of the latest operation end dates [ART 00]. This algorithm is based on a forward traversal of graph G which modifies at each iteration an insertion interval containing a maximal set of arcs, called the *insertion cut*, by replacing resource type arcs going to a node by resource type arcs issued from the same node. For each insertion cut, the algorithm removes useless arcs for allocation, in order to increase the time window while conserving a sufficient number of resource units. A heuristic based on this principle was proposed for the consideration of setup activity which may occur following the inserted operation.

These propagation and insertion methods are used for real-time decision support in order to prepare for hazards, but also during off-line scheduling in iterative improvement methods for the initially generated sequence of groups.

12.6. Off-line scheduling method

12.6.1. *Gradual construction of a feasible initial sequence of groups*

A *feasible* sequence of groups is a sequence respecting all constraints of the problem, except possibly for job delivery dates, which distinguishes it from an *eligible* sequence which must also respect these delivery dates. A feasible initial sequence of groups and corresponding graph of groups can be constructed gradually, using a method presented in detail in [ART 97], by adding a new production operation to a partial sequence at each step. This operation is selected within a series of candidate operations with the help of a priority rule. Please see Chapter 6 or [BLA 82] for states of the art concerning priority rules in job shop scheduling and [KUR 82] in multi-project scheduling.

The difference from traditional methods is that the selected operation position in the partial sequence is obtained by applying the insertion algorithm. The priority rule used must reach a compromise between three objectives sorted in a preferential sequence. The first objective is meeting the delivery dates of jobs to be fulfilled. In order to do this, a priority rule based on the margin of operations is used. The second objective, linked to resource setup activity, carries out technological groupings, i.e. maximum sequence of production operations from the same family by using the same main resources in order to avoid unproductive setup times. We should note that there may be incompatibility between these two objectives. The third objective, linked to the real-time decision support phase, is the maximization of the number of solutions proposed. In the interchangeable operation group context, the idea is to include as many operations as possible in each group. This objective can also turn out to be incompatible with delivery date satisfaction since interchangeability adds constraints to start operation dates which can lead to job delays.

The accomplishment of the delay/setup and delay/interchangeability compromise uses constraint analysis (see Chapter 5). In fact, to reach the delay/setup compromise for a given iteration of the gradual construction, an estimated latest end date is calculated for each production operation, considering the partial sequence generated and the corresponding job delay. In case of conflict between the selection of an urgent operation (relative to its margin) and selection of an operation to complete the technological grouping or to increase the number of operations of a group, the priority rule proposed selects the second operation insofar as this selection is compatible with the estimated latest end date of more urgent operations. Similarly,

when a production operation generating no setup is selected by this priority rule, its insertion in the already existing group (when appropriate) over all resources involved is only carried out with respect to latest estimated end dates for operations already present in the group. If interchangeability constraints cause the violation of one of these dates or if the selected production operation generates a setup activity, a new group not containing this operation is then created (possibly preceded by the different setup operations required).

12.6.2. *Search for eligibility by iterative improvement of the sequence*

Despite constraint analysis carried out during generation, the priority rule used cannot guarantee respect for all job due dates. In the case where a delay is indicated by a positive potential associated with node H of the graph, a group sequence iterative improvement algorithm is then executed based on a neighborhood search method. This method identifies the critical path in the graph representing the current set of solutions, i.e. the set of operations the sequence of which makes this potential positive. From the current set of solutions, a neighboring set of solutions is obtained by deleting a critical operation from the graph and by re-inserting it in another position with the help of the polynomial insertion algorithm presented in section 12.5.3. Selecting operations to move and ending algorithm conditions are performed according to principles from the tabu search method (see [GLO 93, NOW 96] and Chapter 3 on metaheuristics), making it possible to temporarily accept eligibility deteriorations to get rid of a possible local *optimum*. The proposed tabu search method, which is an extension of that presented in [DAU 98], is described in detail in [ART 97]. It has obtained good results in resource-constrained project scheduling problems [ART 99] which integrate the main constraints retained in this chapter.

12.7. Real-time scheduling method, interactive decision support system

The real-time scheduling method proposed is implemented by an interactive decision support system. Section 12.7.1 first presents how real-time information (events, decisions) is retrieved in this system from the workshop and how decision support information is sent to the different decision centers. Among this decision support information, section 12.7.2 puts emphasis on eligibility control indicators for the current sequence of groups, since its respect remains the main objective of real-time scheduling. In fact, decision support itself is different whether we are in an eligible context or not. As long as the current sequence of groups is eligible, decision support procedures presented in section 12.7.3 are intended to help in making operation start decisions favoring conservation of this eligibility. When the sequence becomes non-eligible, decision support procedures presented in section 12.7.4 are intended to retrieve eligibility, prior to continuing to use the sequence of

groups for decision support. Finally, section 12.7.5 presents the possibility of testing decisions outside of the planned context and thus extending the use of the proposed decision support system for negotiation between decision centers.

12.7.1. *Decision support system organization*

Interactions between the decision support system for real time scheduling and its environment are represented in Figure 12.3. The system stores the current workshop state in the form of a Petri net discussed in section 12.4, which makes it possible to know all possible decisions and events at any given time. It also stores the current set of off-line schedules in the form of a potential-task graph presented in section 12.5.2, which makes it possible to know all projected start decisions, the expected set of end of execution events and eligibility of the current set of schedules. In the information technology context, the decision support system is connected to a series of identified decision centers located in the workshop and in scheduling offices by an inter-process communication mechanism with the help of message queues.

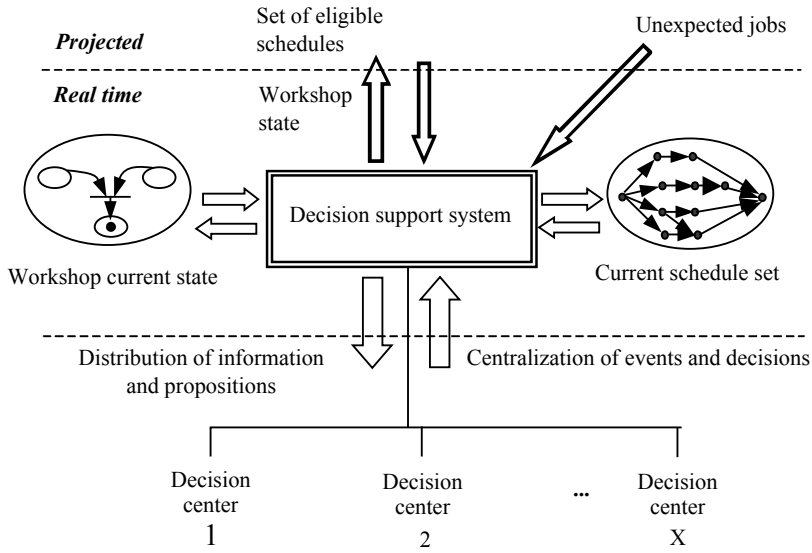


Figure 12.3. *Structure of the decision support system*

A decision center groups a set of entities in a logical link. A decision center can, for example, be made up of a single resource, or a series of resources grouped in a "section" controlled by a single decider. A more "project" orientated global decision center, can group a series of jobs and the different resources required to execute

these jobs. Messages sent from a decision center to the decision support system, constituting requests, are stored in a message queue and are processed in sequence according to the FIFO rule. Messages sent from the system to a center are responses to these requests.

The general algorithm of the decision support system is the following. *req* is a message sent by decision center *X* read by the system in the queue at date *t*. If *req* is "simply" an information request, this information is built and sent to center *X*. For each request, the system identifies entities (operations, resources) involved and sends information (new states, margins) for these entities. If *req* is an event or decision declaration, the system verifies validity of *req* by testing if a corresponding transition can be fired in the current Petri net state. If the test is positive, the transition is fired and Petri net marking is updated. If the test is negative, invalid request information is sent to the decision center with comments on the error. In the case of a valid request, if the decision or event requires it, the potential task graph is updated and dates associated with operations are refreshed according to date *t*. For example, following the end of operation commitment, the corresponding node is deleted from the graph, and based on the event date, the possible modification of operation start dates is propagated in the graph. Finally, information required following the occurrence of the event or decision *req* is sent back to center *X* which transmitted *req*. We should mention that one of the implementation problems with this type of system is workshop division into an adequate number of decision centers.

12.7.2. *Eligibility control*

In general, the response to an expected event or to a decision, such as the end of operation execution on a set of resources, is made up of the new state of the resource(s) involved, the list of available or approaching operations allocated to this (these) resource(s) and the margins of each operations. To reduce the information that a single user has to manipulate, operations with a blank state will not be displayed, except if the operation is located in the leading group of a series of resources (i.e. the group with no "resource" type predecessor in the projected sequence at decision time *t*).

Margins of (production or setup) operations, either in-process or located in a leading group, are sufficient enough to inform decision makers on global eligibility without the need for propagation. In fact, calculating the margin of a production or setup operation *o* only involves the earliest start date of the operation calculated from *t* (denoted as $r_o(t)$), duration p_o and latest end date d_o of the operation calculated at the end of the generation procedure. In real time, there are two margins associated with a single operation. *Net margin* $mp_o(t)$ of a production or setup

operation *o* gives information about eligibility of the sequence in relation to this operation regardless of other operations in the same group. Its expression is:

$$mp_o(t) = d_o - p_o - r_o(t) \quad [12.1]$$

If $|g_{ij}| > 1$, *group margin* $mg_{ij}(t)$ of a production operation $o(i, j)$ gives information on eligibility of the sequence when *o(i, j)* is executed first in its group g_{ij}, to preserve all permutations in g_{ij}. Its expression is:

$$mg_{ij}(t) = min\{d_{xy} | o(x, y) \in g_{ij},\ o(x, y) \neq o(i, j)\} - \Sigma\{p_{xy} | o(x, y) \in g_{ij}\} - r_{ij}(t) \quad [12.2]$$

The minimum of two margins [12.1] and [12.2] is called *sequential free margin* of the operation [THO 80]. A group sequence is eligible if and only if the sequential free margin of all operations located in a leading group is positive or zero. Beyond global eligibility, each sequential free margin gives information on degrees of freedom concerning start operation date (for production or setup operations) as well as on sequential degrees of freedom within group g_{ij} (only for production operations). For example, if the net margin of an operation is positive and its group margin is negative, this means that permutation with the operation processed first in its group is no longer eligible. Nevertheless, if another group operation *g* exists with a group margin that is positive, then the permutation with this operation processed first in the sequence retains eligibility. When an operation sees its sequential free margin become negative, then at least one job will end after its due date if start decisions are taken in the order suggested by the off-line schedule. In order to have a precise list of late jobs, propagation (see section 12.5.3) is necessary for calculating the earliest projected end date of each job.

In addition, in response to an unexpected event declaration (disruption) such as resource breakdown with an estimated duration or more generally following an event making certain margins negative, the system sends back the same information as before, completed by the list of jobs that the hazard potentially delays beyond their delivery date.

12.7.3. *Decision support in an eligible sequence context*

Information given in the context where the current sequence of groups is eligible makes it possible to help in choosing the next start decisions in the decision center involved. If no displayed margin is negative (if no job is late), then eligibility is respected and the system suggests the selection of start decisions within the set of projected solutions, even in the presence of disruptions. Operations in the center involved are displayed according to a total sequence corresponding to a preferential commitment sequence, including production operations belonging to the same group

of interchangeable operations *g*. We can in fact show that the best solution to save degrees of freedom associated with *g* is to start production operations of *g* in the decreasing order of their sequential free margin [THO 80]. However, this sequence is only a proposal and operations of the same group can be executed in any order.

12.7.4. *Decision support for retrieving eligibility*

When a margin becomes negative following a disruption or a delay of the occurrence of a projected decision or an expected event, the set of schedules described by the projected sequence of groups becomes ineligible and strict respect of the planned context cannot improve the situation. On the other hand, if detected delays of jobs are not "too" significant, we can presume that local modifications consisting of making one or more decision(s) outside of the planned context may restore eligibility.

The system then tests a series of possible decisions depending on the hazard which caused the problem. If an operation is blocked when it uses resources that work well, the system tests the start of other operations on these resources. This start is possible as long the blocked operation is interrupted. If a resource fails, the system tests the start of waiting operations on a replacement resource. In all cases, these tests are carried out on the potential-task graph in order to evaluate the impact of alternative decisions on eligibility. The polynomial insertion algorithm presented in section 12.5.3 is the basic tool for these local modifications which follow the principle of the iterative improvement procedure presented in section 12.6.2, i.e. reducing the length of the critical path, by deletion and reinsertion of critical operations while retaining the relative sequence of groups.

This local character makes it possible to retain most of the anticipations that could have been carried out following the known projected schedule and ensures schedule stability. In return, if disruptions are too large, a return to the global generation of a set of solutions may be necessary. In practice, this capacity of two level reactions is well adapted to reality in a workshop.

12.7.5. *Decision and negotiation support between decision centers outside the planned context*

Even though the decision maker has some level of freedom proposed by the system, he may wish to consider decisions other than the ones proposed to him. The system must therefore let him test the consequences of this type of decision, which are outside of the planned context. The insertion algorithm presented in section 12.5.3 makes it possible to quickly test the impact on eligibility:

– of the start of an operation not located in a leading group;

– of the modification of the projected execution mode for an operation;

– of the insertion (one operation at a time) of an unexpected job not considered during off-line scheduling.

Even if, within a given decision center, this possibility gives more flexibility to the decision maker, it also provides help in negotiating between the various decision centers as the following scenarios show:

– in the multi-resource context, following a breakdown (absence of operators for example), center A may require replacement with certain resources from center B. The use of the system can prove that unexpected mobilization of these resources will not disrupt eligibility of sets of schedules. In this way, decision makers from center B can more readily accept that "their" resources be used outside of the context initially planned;

– when receiving an urgent order, the sales department can test the insertion impact of the associated new job in the set of schedules and see its repercussions on delays of other jobs (and thus of other clients), which provides elements in the negotiation between the client and sales department, as well as between the sales department and production;

– concerning setup times, section managers can be tempted to maximize technological groupings whereas scheduling managers are mostly concerned by delays. We can see the advantage of the possibility of testing the eligibility of start decisions outside of the planned context, such as the unexpected insertion of an operation of the same family, enabling these decision makers to measure the consequences of their own preferences over the global performance of the workshop.

12.8. Conclusion

Company environments and market demand require increased performance of product flow management during production. This management involves a certain number of functions among which workshop scheduling is becoming increasingly important. It is in fact this function which manages all operations to be performed to obtain the required final product. In order to achieve this, numerous disruptions can occur, requiring a dynamic vision of the scheduling problem. In response to this objective, a trend is appearing directing research toward real-time scheduling, for taking into account the robustness and flexibility necessary for this real time approach. Studies in this field are still limited and it seemed useful to present in this book a methodology which is advanced enough to enable the definition and implementation of a real interactive decision support system for real-time

scheduling. This methodology, along with its associated methods and tools, made it possible to develop a computer program that is at the core of a commercial software package called ORDO implemented in over sixty companies. Details about this software are available in [THO 88, ROU 98].

However, this only constitutes a specific direction; other approaches need to be explored, as was mentioned in this chapter and in others. Real-time scheduling constitutes an open problem for research, where results will surely contribute to better efficiency activity management systems in the goods or services manufacturing field.

12.9. Bibliography

[ADA 88] ADAMS J., BALAS E. and ZAWACK D., "The shifting bottleneck procedure for job shop scheduling", *Management Science*, vol. 34, no. 3, p. 391-401, 1988.

[ALB 03] ALBERS S., "Online algorithms: a survey", *Mathematical Programming*, vol. 97, p. 3-26, 2003.

[ALO 05] ALOULOU M. A. and PORTMANN M.-C., "An Efficient Proactive-Reactive Scheduling Approach to Hedge Against Shop Floor Disturbances", in G. Kendall, E.K. Burke, S. Petrovic and M. Gendreau, (eds.), *Multidisciplinary Scheduling: Theory and Applications*, Springer, p. 223-246, 2005.

[ART 97] ARTIGUES C., Ordonnancement en temps réel d'ateliers avec temps de préparation des ressources, PhD thesis, Paul Sabatier University, Toulouse, 1997.

[ART 99] ARTIGUES C., ROUBELLAT F. and BILLAUT J.-C., "Characterization of a set of schedules in a resource constrained multiproject scheduling problem with multiple modes", *International Journal of Industrial Engineering, Applications and Practice*, Special Issue On Project Management, vol. 6, no. 2, p. 112-222, 1999.

[ART 00] ARTIGUES C. and ROUBELLAT F., "A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and resource flexibility", *European Journal of Operational Research*, vol. 127, no. 2, p. 297-316, 2000.

[ART 03] 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.

[AYT 05] AYTUG H., LAWLEY M.A., MCKAY K., MOHAN S., UZSOY R., "Executing production schedules in the face of uncertainties: a review and some future directions", *European Journal of Operational Research*, vol. 161, no. 1, p. 86-110, 2005.

[BAL 95] BALAS E., LENSTRA J.K. and VAZACOPOULOS A., "The one machine problem with delayed precedence constraints and its use in job shop scheduling", *Management Science*, vol. 41, no. 1, p. 94-109, 1995.

[BIL 96a] BILLAUT, J.-C. and ROUBELLAT F., "A New Method for Workshop Real-Time Scheduling", *International Journal of Production Research*, vol. 34, no. 6, p. 1555-1579, 1996.

[BIL 96b] BILLAUT J.-C., ROUBELLAT F., "Characterization of a set of schedules in a multiresource context", *Journal of Decision Systems*, vol. 5, no. 1-2, p. 95-109, 1996.

[BLA 82] BLACKSTONE J.H., PHILIPS D.T. and HOGG G.L., "A state of the art survey of dispatching rules for manufacturing job-shop operations", *International Journal of Production Research*, vol. 20, p. 27-45, 1982.

[BLA 96] BLAZEWICZ J., DOMSCHKE W. and PESCH E., "The job-shop scheduling problem: conventional and new solution techniques", *European Journal of Operational Research*, vol. 93, no. 1, p. 1-33, 1996.

[BRI 07] C. BRIAND, H. LA, J. ERSCHLER, "A robust approach for the single machine scheduling problem", *Journal of Scheduling*, vol. 10, no. 3, p. 209-221, 2007.

[CAR 89] CARLIER J. and PINSON E., "An algorithm for solving the job shop problem", *European Journal of Operational Research*, vol. 35, no. 2, p. 164-176, 1989.

[DAN 95] R. L. DANIELS and P. KOUVELIS, "Robust Scheduling to Hedge against Processing Time Uncertainty in Single-Stage Production", *Management Science*, vol. 41, no. 2, p. 363-376, 1995.

[DAU 98] DAUZÈRE-PÉRÈS S., ROUX W. and LASSERRE J.B., "Multi-resource shop scheduling with resource flexibility", *European Journal of Operational Research*, vol. 107, p. 289-305, 1998.

[DEM 77] DEMMOU R., Etude de familles remarquables d'ordonnancement en vue d'une aide à la décision, PhD thesis, Paul Sabatier University, Toulouse, 1977.

[DUB 95] DUBOIS D., FARGIER H. And PRADE H., "Fuzzy constraints in job-shop scheduling", *Journal of Intelligent Manufacturing*, vol. 6, no. 4, p. 215-234, 1995.

[ERS 89] ERSCHLER, J. and ROUBELLAT, F. „An approach for real time scheduling for activities with time and resource constraints" in *Advances in Project Scheduling* Slowinski, R. and Weglarz, J., (eds.), Elsevier, 1989.

[GAV 65] GAVETT J.W., "Three heuristic rules for sequencing jobs to a single production facility", *Management Science*, vol. 11, no. 8, p. 166-176, 1965.

[GIA 88] GIARD V., *Gestion de la production*, 2nd edition, Economica, 1988.

[GLO 93] GLOVER F., TAILLARD E. and DE WERRA D., "A user's guide to tabu search", *Annals of Operations Research*, vol. 41, p. 3-28, 1993.

[GOT 93] GOThA, Groupe de Recherche en Ordonnancement Théorique et Appliqué (Carlier J., Chrétienne P., Erschler J., Hanen C., Lopez P., Munier A., Pinson E., Portmann M.-C., Prins C., Proust C. and Villon P.), "Les problèmes d'ordonnancement", *Recherche Opérationnelle/Operations Research*, vol. 27, no. 1, p. 77-150, 1993.

[GUI 91] GUINET A., "Textile production systems: a succession of non-identical parallel processor shops", *Journal of Operational Research Society,* vol. 42, no. 8, p. 665-671, 1991.

[HER 98] HERROELEN W., DEMEULEMEESTER E. and DE REYCK B., "A classification scheme for project scheduling", in *Project Scheduling: Recent Models, Algorithms and Applications*, p. 1-26, J. Weglarz (ed.), Kluwer, Amsterdam, 1998.

[HER 05] HERROELEN W., LEUS R., "Project scheduling under uncertainty: survey and research potentials", *European Journal of Operational Research*, vol. 165, no. 2, p. 289-306, 2005.

[KOL 95] KOLISH R., *Project Scheduling Under Resource Constraints*, Physica-Verlag, 1995.

[KUR 82] KURTULUS I.S. and DAVIS E.W., "Multi-project scheduling: categorization of heuristic rules performance", *Management Science*, vol. 28, no. 2, p. 161-172, 1982.

[LEG 92] LE GALL A., ROUBELLAT F., "Caractérisation d'un ensemble d'ordonnancements avec contraintes de ressources de type cumulative", *Automatique-productique informatique industrielle*, vol. 26, no. 5-6, p. 515-535, 1992.

[MAR 96] MARTIN P. and SHMOYS D., "A new approach to compute optimal schedules for the job-shop scheduling problem", *5th International IPCO Conference*, p. 389-403, 1996.

[MAS 02] MASON S.J., FOWLER J.W., and CARLYLE. W.M., "A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops", *Journal of Scheduling*, vol. 5, no. 3, p. 247-262, 2002.

[MEE 96] MEESTER G. J., Multi-resource shop floor scheduling, PhD thesis, University of Twente, Enschede, The Netherlands, 1996.

[MON 93] MONMA C.L. and POTTS C.N., "Analysis of heuristics for preemptive parallel machine scheduling with batch setup times", *Operations Research*, vol. 41, no. 5, p. 981-993, 1993.

[NOW 96] NOWICKI E. and SMUTNICKI C., "A fast tabu search algorithm for the job-shop problem", *Management Science*, vol. 42, no. 6, p. 797-813, 1996.

[OVA 94] OVACIK I. M. and UZSOY R., "Exploiting shop floor status information to schedule complex job shops", *Journal of Manufacturing Systems*, vol. 13, p. 73-84, 1994.

[PRO 91] PROUST C., GUPTA J.N.D. and DESCHAMPS V., "Flowshop scheduling with set-up, processing and removal times separated", *International Journal of Production Research*, vol. 29, no. 3, p. 479-493, 1991.

[ROU 98] ROUBELLAT F., ARTIGUES C., VILLAUMIE M., "Un système interactif d'aide à la décision pour l'ordonnancement en temps réel d'atelier", *La Revue Française de Gestion Industrielle*, 17(4):5-20, 1998.

[ROY 70] ROY B., *Algèbre moderne et théorie des graphes*, volume II, Dunod, 1970.

[SCH 98] SCHUTTEN J.M.J., "Practical job shop scheduling", *Annals of Operations Research*, vol. 83, no. 0, 1998.

[SHI 86] SHINGO S., *Maîtrise de la production et méthode Kanban*, Les Editions d'Organisation, 1986.

[SIL 94] SILLY M., "Un algorithme d'ordonnancement de tâches sporadiques pour les systèmes temps-réel", *APII*, vol. 28, no. 2, p. 179-205, 1994.

[SKU 05] SKUTELLA M., UETZ M., "Stochastic machine scheduling with precedence constraints", *SIAM Journal on Computing*, vol. 34, no. 4, p. 788-802, 2005.

[STA 95] STANKOVIC J. A., SPURI M., DI NATALE M. and BUTTAZZO G. C., "Implications of Classical Scheduling Results for Real-Time Systems", *Computer*, vol. 28, no. 6, p. 16-25, 1995.

[THO 80] THOMAS V., Aide à la décision pour l'ordonnancement en temps réel, PhD thesis, Paul Sabatier University, Toulouse, 1980.

[THO 88] THOMAS V., ROUBELLAT F., Une méthode et un logiciel pour l'ordonnancement en temps réel d'ateliers, Automatique-productique informatique industrielle, vol. 22, no. 5, p. 419-438, 1988.

[VIE 03] G. E. VIEIRA, J. W. HERRMANN, E. LIN, "Rescheduling manufacturing systems: A framework of strategies, policies, and methods", *Journal of Scheduling*, vol. 6, no. 1, p. 39-62, 2003.

[WU 99] WU S.D., BYEON E. and STORER R.H., "A graph-theoretic decomposition of job shop scheduling problems to achieve scheduling robustness", *Operations Research*, vol. 47, p. 113–124, 1999.

[XIE 89] XIE, X., "Real-time scheduling and routing for flexible manufacturing systems with unreliable machines", *Recherche Opérationnelle/Operations Research*, vol. 23, no. 4, p. 355-374, 1989.

This page intentionally left blank

List of Authors

Christian ARTIGUES
LAAS-CNRS
University of Toulouse, France

Pierre BAPTISTE
MAGI - GERAD
École Polytechnique de Montréal, Quebec, Canada

Gérard BEL
ONERA
Toulouse, France

Jean-Charles BILLAUT
Laboratoire d'Informatique
Ecole Polytechnique, University of Tours, France

Christelle BLOCH
LIFC/University of Franche-Comté
Montbéliard, France

Hervé CAMUS
LAIL
Ecole Centrale de Lille, France

Jacques CARLIER
HeuDiaSyc, CNRS
Compiègne University of Technology, France

Jean-Bernard CAVAILLÉ
ONERA
Toulouse, France

Daniel COSTA
Supply Chain
Nestlé, Italy

Didier DUBOIS
IRIT-UPS, CNRS
University of Toulouse, France

Patrick ESQUIROL
LAAS-CNRS
University of Toulouse, France

Hélène FARGIER
IRIT-UPS, CNRS
University of Toulouse, France

Philippe FORTEMPS
Faculty of Engineering, Mons, Belgium

Jean-Claude GENTINA
LAIL
Ecole Centrale de Lille, France

Alain HERTZ
MAGI - GERAD
École Polytechnique de Montréal, Quebec, Canada

Marie-José HUGUET
LAAS-CNRS
University of Toulouse, France

Ouajdi KORBAA
SOIE
University of Sousse, Tunisia

Pierre LOPEZ
LAAS-CNRS
University of Toulouse, France

Marie-Ange MANIER
Belfort-Montbéliard University of Technology, France

Emmanuel NÉRON
Laboratoire d'Informatique
Ecole Polytechnique, University of Tours, France

Antoine OLIVER
Laboratoire d'Informatique
Ecole Polytechnique, University of Tours, France

Marie-Claude PORTMANN
LORIA
Ecole des Mines de Nancy, France

Christian PRINS
LOSI
University of Technology of Troyes, France

François ROUBELLAT
LAAS-CNRS
University of Toulouse, France

Christophe VARNIER
FEMTO-ST Institute
ENSMM/University of Franche-Comté, CNRS
Besançon, France

Antony VIGNIER
LORIA
Ecole des Mines de Nancy, France

Marino WIDMER
DIUF
University of Fribourg, Switzerland

This page intentionally left blank

Index

N–O

P

U–W