

Flexibility and Robustness in Scheduling

Flexibility and Robustness in Scheduling

Edited by
Jean-Charles Billaut
Aziz Moukrim
Eric Sanlaville



First published in France in 2005 by Hermes Science/Lavoisier entitled: "Flexibilité et robustesse en ordonnancement"

First published in Great Britain and the United States in 2008 by ISTE Ltd and John Wiley & Sons, Inc.

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms and licenses issued by the CLA. Enquiries concerning reproduction outside these terms should be sent to the publishers at the undermentioned address:

ISTE Ltd
27-37 St George's Road
London SW19 4EU
UK

www.iste.co.uk

© ISTE Ltd, 2008
© LAVOISIER, 2005

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

www.wiley.com

The rights of Jean-Charles Billaut, Aziz Moukrim and Eric Sanlaville to be identified as the authors of this work have been asserted by them in accordance with the Copyright, Designs and Patents Act 1988.

Library of Congress Cataloging-in-Publication Data

[Flexibilité et robustesse en ordonnancement English] Flexibility and robustness in scheduling / Edited by Jean-Charles Billaut, Aziz Moukrim, Eric Sanlaville.

p. cm.

Includes bibliographical references and index.

ISBN 978-1-84821-054-7

I. Production scheduling. I. Billaut, Jean-Charles 1973-. II. Moukrim, Aziz. III. Sanlaville, Eric.
TS157.5.F55 2008
658.5'3--dc22

2008030722

British Library Cataloguing-in-Publication Data

A CIP record for this book is available from the British Library

ISBN: 978-1-84821-054-7

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



Cert no. SGS-COC-2993
www.fsc.org
© 1996 Forest Stewardship Council

Table of Contents

Preface	13
Chapter 1. Introduction to Flexibility and Robustness in Scheduling	15
Jean-Charles BILLAUT, Aziz MOUKRIM and Eric SANLAVILLE	
1.1. Scheduling problems	15
1.1.1. Machine environments	16
1.1.2. Characteristics of tasks	17
1.1.3. Optimality criteria	18
1.2. Background to the study	19
1.3. Uncertainty management	20
1.3.1. Sources of uncertainty	21
1.3.2. Uncertainty of models	22
1.3.3. Possible methods for problem solving	23
1.3.3.1. Full solution process of a scheduling problem with uncertainties	23
1.3.3.2. Proactive approach	24
1.3.3.3. Proactive/reactive approach	24
1.3.3.4. Reactive approach	25
1.4. Flexibility	25
1.5. Robustness	26
1.5.1. Flexibility as a robustness indicator	27
1.5.2. Schedule stability (solution robustness)	28
1.5.3. Stability relatively to a performance criterion (quality robustness)	29
1.5.4. Respect of a fixed performance threshold	30
1.5.5. Deviation measures with respect to the optimum	30
1.5.6. Sensitivity and robustness	31
1.6. Bibliography	31

6 Flexibility and Robustness in Scheduling

Chapter 2. Robustness in Operations Research and Decision Aiding	35
Bernard ROY	
2.1. Overview	35
2.1.1. Robust in OR-DA with meaning?	36
2.1.2. Why the concern for robustness?	37
2.1.3. Plan of the chapter	38
2.2. Where do “vague approximations” and “zones of ignorance” come from? – the concept of version	38
2.2.1. Sources of inaccurate determination, uncertainty and imprecision	38
2.2.2. DAP formulation: the concept of version	40
2.3. Defining some currently used terms	41
2.3.1. Procedures, results and methods	41
2.3.2. Two types of procedures and methods	42
2.3.3. Conclusions relative to a set \hat{R} of results	43
2.4. How to take the robustness concern into consideration	43
2.4.1. What must be robust?	44
2.4.2. What are the conditions for validating robustness?	45
2.4.3. How can we define the set of pairs of procedures and versions to take into account?	46
2.5. Conclusion	47
2.6. Bibliography	47
Chapter 3. The Robustness of Multi-Purpose Machines Workshop Configuration	53
Marie-Laure ESPINOUSE, Mireille JACOMINO and André ROSSI	
3.1. Introduction	53
3.2. Problem presentation	53
3.2.1. Modeling the workshop	54
3.2.1.1. Production resources	54
3.2.1.2. Modeling the workshop demand	55
3.2.2. Modeling disturbances on the data	55
3.2.3. Performance versus robustness: load balance and stability radius .	57
3.2.3.1. Performance criterion for a configuration	57
3.2.3.2. Robustness	57
3.3. Performance measurement	57
3.3.1. Stage one: minimizing the maximum completion time	57
3.3.2. Computing a production plan minimizing machine workload . .	59
3.3.3. The particular case of uniform machines	60
3.4. Robustness evaluation	61
3.4.1. Finding the demands for which the production plan is balanced .	61
3.4.2. Stability radius	64
3.4.3. Graphic representation	65

3.5. Extension: reconfiguration problem	68
3.5.1. Consequence of adding a qualification to the matrix Q	68
3.5.2. Theoretical example	69
3.5.3. Industrial example	70
3.6. Conclusion and perspectives	70
3.7. Bibliography	71
Chapter 4. Sensitivity Analysis for One and m Machines	73
Amine MAHJOUB, Aziz MOUKRIM, Christophe RAPINE and Eric SANLAVILLE	
4.1. Sensitivity analysis	74
4.2. Single machine problems	78
4.2.1. Some analysis from the literature	78
4.2.2. Machine initial unavailability for $1 \parallel \sum U_j$	79
4.2.2.1. Problem presentation	79
4.2.2.2. Sensitivity of the HM algorithm	80
4.2.2.3. Hypotheses and notations	80
4.2.2.4. The two scenario case	81
4.3. m -machine problems without communication delays	83
4.3.1. Parametric analysis	83
4.3.2. Example of global analysis: $Pm \parallel \sum C_j$	85
4.4. The m -machine problems with communication delays	87
4.4.1. Notations and definitions	88
4.4.2. The two-machine case	90
4.4.3. The m -machine case	92
4.4.3.1. Some results in a deterministic setting	92
4.4.3.2. Framework for sensitivity analysis	93
4.4.3.3. Stability studies	93
4.4.3.4. Sensitivity bounds	94
4.5. Conclusion	95
4.6. Bibliography	96
Chapter 5. Service Level in Scheduling	99
Stéphane DAUZÈRE-PÉRÈS, Philippe CASTAGLIOLA and Chams LAHLOU	
5.1. Introduction	99
5.2. Motivations	101
5.3. Optimization of the service level: application to the flow-shop problem	103
5.3.1. Criteria computation	103
5.3.2. Processing time generation	104
5.3.3. Experimental results	106
5.4. Computation of a schedule service level	109
5.4.1. Introduction	110
5.4.2. FORM (First Order Reliability Method)	111
5.4.3. FORM vs Monte Carlo	112

8 Flexibility and Robustness in Scheduling

5.5. Conclusions	118
5.6. Bibliography	119
Chapter 6. Metaheuristics for Robust Planning and Scheduling	123
Marc SEVAUX, Kenneth SØRENSEN and Yann LE QUÉRÉ	
6.1. Introduction	123
6.2. A general framework for metaheuristic robust optimization	124
6.2.1. General considerations	124
6.2.2. An example using a genetic algorithm	126
6.3. Single-machine scheduling application	127
6.3.1. Minimizing the number of late jobs on a single machine	127
6.3.2. Uncertainty of deliveries	129
6.3.2.1. Considered problem	129
6.3.2.2. Robust evaluation function	129
6.3.3. Results	130
6.4. Application to the planning of maintenance tasks	132
6.4.1. SNCF maintenance problem	133
6.4.2. Uncertainties of an operational factory	134
6.4.3. A robust schedule	135
6.4.3.1. Variations of the unexpected factors	137
6.5. Conclusions and perspectives	139
6.6. Bibliography	140
Chapter 7. Metaheuristics and Performance Evaluation Models for the Stochastic Permutation Flow-Shop Scheduling Problem	143
Michel GOURGAND, Nathalie GRANGEON and Sylvie NORRE	
7.1. Problem presentation	144
7.2. Performance evaluation problem	147
7.2.1. Markovian analysis	147
7.2.2. Monte Carlo simulation	153
7.3. Scheduling problem	155
7.3.1. Comparison of two schedules	156
7.3.2. Stochastic descent for the minimization in expectation	157
7.3.3. Inhomogenous simulated annealing for the minimization in expectation	157
7.3.4. Kangaroo algorithm for the minimization in expectation	159
7.3.5. Neighboring systems	161
7.4. Computational experiment	161
7.4.1. Exponential distribution	162
7.4.2. Uniform distribution function	164
7.4.3. Normal distribution function	167
7.5. Conclusion	167
7.6. Bibliography	168

Chapter 8. Resource Allocation for the Construction of Robust Project Schedules	171
Christian ARTIGUES, Roel LEUS and Willy HERROELEN	
8.1. Introduction	171
8.2. Resource allocation and resource flows	173
8.2.1. Definitions and notation	173
8.2.2. Resource flow networks	174
8.2.3. A greedy method for obtaining a feasible flow	176
8.2.4. Reactions to disruptions	176
8.3. A branch-and-bound procedure for resource allocation	178
8.3.1. Activity duration disruptions and stability	178
8.3.2. Problem statement and branching scheme	179
8.3.3. Details of the branch-and-bound algorithm	180
8.3.4. Testing for the existence of a feasible flow	182
8.3.5. Branching rules	183
8.3.6. Computational experiments	184
8.3.6.1. Experimental setup	184
8.3.6.2. Branching schemes	185
8.3.6.3. Comparison with the greedy heuristic	187
8.4. A polynomial algorithm for activity insertion	187
8.4.1. Insertion problem formulation	188
8.4.2. Evaluation of a feasible insertion	189
8.4.3. Insertion feasibility conditions	190
8.4.4. Sufficient insertions and insertion cuts	191
8.4.5. Insertion dominance conditions	192
8.4.6. An algorithm for enumerating dominant sufficient insertions	193
8.4.7. Experimental results	193
8.5. Conclusion	194
8.6. Bibliography	195
Chapter 9. Constraint-based Approaches for Robust Scheduling	199
Cyril BRIAND, Marie-José HUGUET, Hoang Trung LA and Pierre LOPEZ	
9.1. Introduction	199
9.2. Necessary/sufficient/dominant conditions and partial orders	200
9.3. Interval structures, tops, bases and pyramids	201
9.4. Necessary conditions for a generic approach to robust scheduling	202
9.4.1. Introduction	202
9.4.2. Scheduling problems under consideration	204
9.4.3. Necessary feasibility conditions	205
9.4.4. Propagation mechanisms	206
9.4.4.1. Time constraint propagation	206
9.4.4.2. Resource constraint propagation	207

9.4.5. Interval structures for propagation	208
9.4.5.1. Rank-interval based structures	208
9.4.5.2. Task-interval based structures	210
9.4.6. Discussion	212
9.5. Using dominance conditions or sufficient conditions	213
9.5.1. The single machine scheduling problem	213
9.5.2. The two-machine flow-shop problem	217
9.5.3. Future prospects	221
9.6. Conclusion	222
9.7. Bibliography	222

Chapter 10. Scheduling Operation Groups: A Multicriteria Approach to Provide Sequential Flexibility 227

Carl ESSWEIN, Jean-Charles BILLAUT and Christian ARTIGUES

10.1. Introduction	227
10.2. Groups of permutable operations	228
10.2.1. History, principles and definitions	228
10.2.2. Representation and evaluation	230
10.2.2.1. Earliest start time computation	232
10.2.2.2. Latest completion time computation	234
10.2.2.3. Quality of a group schedule	234
10.3. The ORABAID approach	235
10.3.1. The proactive phase: searching for a feasible and acceptable group schedule	235
10.3.1.1. Construction of a feasible group schedule	236
10.3.1.2. Searching for acceptability of the group schedule	237
10.3.1.3. Increasing the group schedule flexibility	237
10.3.2. The reactive phase: real-time decision aid	237
10.3.3. Some conclusions about ORABAID	238
10.4. AMORFE, a multicriteria approach	238
10.4.1. Flexibility evaluation of a group schedule	239
10.4.2. Evaluation of the quality of a group schedule	240
10.4.3. Some considerations about the objective function definition	241
10.4.4. Quality guarantee in the best case	243
10.4.4.1. Advantages	243
10.4.4.2. Respect for quality in the best case	243
10.5. Application to several scheduling problems	244
10.6. Conclusion	246
10.7. Bibliography	246

Chapter 11. A Flexible Proactive-Reactive Approach: The Case of an Assembly Workshop 249

Mohamed Ali ALOULOU and Marie-Claude PORTMANN

11.1. Context	249
-------------------------	-----

11.2. Definition of the control model	251
11.2.1. Definition of the problem and its environment	251
11.2.2. Definition of a solution to the problem	251
11.2.3. Definition of the solution quality	252
11.2.3.1. Preliminary example	252
11.2.3.2. Performance of a solution	253
11.2.3.3. Flexibility of a solution	255
11.3. Proactive algorithm	256
11.3.1. General schema of the proposed genetic algorithm	256
11.3.2. Selection and strategy of reproduction	258
11.3.3. Coding of a solution	258
11.3.4. Crossover operator	258
11.3.5. Mutation operator	259
11.4. Reactive algorithm	260
11.4.1. Functions of the reactive algorithm	260
11.4.2. Reactive algorithms in the absence of disruptions	261
11.4.2.1. <i>A posteriori</i> quality measures	261
11.4.2.2. Proposed algorithms	263
11.4.3. Reactive algorithm with disruptions	264
11.5. Experiments and validation	264
11.6. Extensions and conclusions	265
11.7. Bibliography	266

Chapter 12. Stabilization for Parallel Applications 269

Amine MAHJOUB, Jonathan E. PECERO SÁNCHEZ and Denis TRYSTRAM

12.1. Introduction	270
12.2. Parallel systems and scheduling	270
12.2.1. Actual parallel systems	270
12.2.2. Definitions and notations	271
12.2.3. Motivating example	273
12.3. Overview of different existing approaches	275
12.4. The stabilization approach	276
12.4.1. Stabilization in processing computing	276
12.4.2. Example	278
12.4.3. Stabilization process	280
12.5. Two directions for stabilization	280
12.5.1. The PRCP* algorithm	281
12.5.2. Strong stabilization	283
12.6. An intrinsically stable algorithm	286
12.6.1. Convex clustering	286
12.6.2. Stability analysis of convex clustering	290
12.7. Experiments	293
12.7.1. Impact of disturbances in the schedules of the three algorithms .	294

12 Flexibility and Robustness in Scheduling

12.7.2. Influence of the initial schedule in the stabilization process	295
12.7.3. Comparison of the schedules with and without stabilization	297
12.7.4. Test 1 – comparison for Winkler graphs	297
12.7.5. Test 2 – comparison for layer graphs	298
12.8. Conclusion	299
12.9. Bibliography	300

Chapter 13. Contribution to a Proactive/Reactive Control of Time Critical

Systems

Pascal AYGALINC, Soizick CALVEZ and Patrice BONHOMME

13.1. Introduction	303
13.2. Static problem definition	305
13.2.1. Autonomous Petri nets (APN)	306
13.2.2. p-time PNs	307
13.3. Step 1: computing a feasible sequencing family	311
13.4. Step 2: dynamic phase	317
13.4.1. Temporal flexibility	317
13.4.2. Temporal flexibility and sequential flexibility	319
13.4.2.1. Partial order in performance evaluation	320
13.4.2.2. Partial order in proactive/reactive control	322
13.5. Restrictions due to p-time PNs	323
13.6. Bibliography	325

Chapter 14. Small Perturbations on Some NP-Complete Scheduling

Problems

Christophe PICOULEAU

14.1. Introduction	327
14.2. Problem definitions	328
14.2.1. Sequencing with release times and deadlines	328
14.2.2. Multiprocessor scheduling	329
14.2.3. Unit execution times scheduling	330
14.2.4. Scheduling unit execution times with unit communication times	331
14.3. NP-completeness results	332
14.4. Conclusion	340
14.5. Bibliography	340

List of Authors

341

Index

347

Preface

This book is about scheduling under uncertainties. However, the problems concern the whole domain of decision aid. Of course, the question of decision aid under unexpected events or uncertainties is not new, but a recent awareness has come on the necessity to define specific models. This awareness has lead to research activities in various domains like location, communication or transportation network design, supply chain management, industrial planing and – of course – scheduling problems.

In Spring 2000, some members of the “GOThA” group (a French working group on “Theoretical scheduling and applications”) decided to create a sub-group working in the field of “flexibility”. It seemed convenient to gather the persons interested by the question: “*how do we schedule under uncertainties?*”. The success of this initiative was a surprise for their promoters themselves. In France only, among ten research teams were working on this problem. These teams wanted to communicate ideas, to unify the terminology, to exchange references. After multiple meetings during 2003, this group became a project “*Scheduling with flexibility and robustness*” among an official structure, the GDR-CNRS on Operations Research. The book *Flexibilité et Robustesse en Ordonnancement* published by Hermès in 2005 was the first conclusion of this project. This book is a revised version of this title.

The outline of the book is the following. The two first chapters are introductory. The first one introduces the problem, the main concepts and basic definitions. The second chapter is written by Bernard Roy, who examines the concept of robustness in the more general framework of decision aid. Subsequent chapters correspond to the specialties of several research teams. They can be organized according to the resolution approach or the application field. Each chapter presents a state-of-the-art survey related to its field.

Chapters 3 to 8 (5 to 8 with probabilistic hypotheses) consider that all the decisions have been taken before starting the schedule. In the approach of Chapters 9 to 13 most of the decisions are taken during the execution of the schedule. The last chapter considers on-line re-optimization.

Scheduling theory concerns several fields. Chapters 3, 5, 7, 10, 11 and 13 consider shop scheduling problems, whereas Chapters 6 and 8 consider project scheduling problems and Chapters 4 and 12 consider parallel computing. Chapters 9 and 14 do not consider a particular application field. But frontiers are sometimes thin.

We are very happy with this English version and hope that it will interest numerous researchers and scheduling practitioners.

Jean-Charles BILLAUT

Aziz MOUKRIM

Eric SANLAVILLE

Chapter 1

Introduction to Flexibility and Robustness in Scheduling

1.1. Scheduling problems

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

- Industrial production systems: problems may need to be solved simultaneously in machine scheduling and vehicle dispatching (automated guided systems, robotic cells, hoist scheduling problems), in workshop layout problems or supply chain management problems.
- Computer systems: for example, to make full use of the processing power provided by parallel machines or when scheduling tasks with resource constraints in real-time environments.
- Administrative systems: appointment scheduling in health care sector, general resource assignment, timetabling, etc.
- Transportation systems: vehicle routing problems, traveling salesman problems, etc.

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

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

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

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

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

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

1.1.1. Machine environments

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

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

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

- If $\alpha_1 = P$, the machines are identical: $p_{ij} = p_j$ for any machine M_i .

- If $\alpha_1 = Q$, the machines are uniform: $p_{ij} = p_j/s_i$ where s_i is the processing speed of machine M_i .

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

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

- Flow-shop. Each job consists of m operations and the order of execution on different machines is the same for each job. In this case $\alpha_1 = F$.

- Job-shop. The number of operations is not necessarily the same for each job, and every job has its own order of execution on the machines. In this case $\alpha_1 = J$.

- Open shop. This is the least constrained shop scheduling problem. The number of operations is not necessarily the same for each job, and the order of execution on the machines is completely free. In this case $\alpha_1 = O$.

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

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

1.1.2. Characteristics of tasks

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

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

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

performed on two different processors (or machines), c_{jk} corresponds to the shortest delay between the end of T_j and the beginning of T_k , otherwise this delay is zero.

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

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

1.1.3. Optimality criteria

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

- end date of the task T_j or job J_j , or completion time, noted C_j ;
- lateness $L_j = C_j - d_j$ or tardiness $T_j = \max\{0, C_j - d_j\}$;
- unit penalty $U_j = 0$ if $C_j \leq d_j$, otherwise $U_j = 1$;
- flow time $F_j = C_j - r_j$.

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

- the duration of the schedule or *makespan* is the function $C_{\max} = \max_{1 \leq j \leq n} C_j$;
- the weighted number of late tasks is the function $\sum_{j=1}^n w_j U_j$.

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

We conclude this section by defining three scheduling classes:

- a schedule is said to be *semi-active* if no task can be performed earlier without changing the order of execution or violating the constraints;
- a schedule is said to be *active* if no task can be performed earlier without violating the constraints;
- a schedule is said to be *without delay* if at any time t resources are not present in sufficient quantity to start an available job processed later in the schedule.

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



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

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

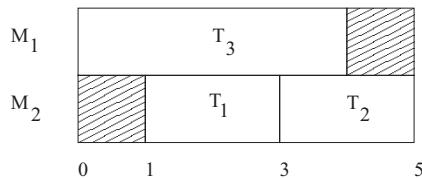


Figure 1.2. An example of an active schedule that is not delay-free

1.2. Background to the study

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

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

- Classical deterministic methods, which consider that the data are deterministic and that the machine environment is relatively simple (disjunctive resources, possibly in multiple copies: see section 1.1.1). Some traditional constraints are taken into account (precedence constraints, release dates, due dates, preemption, etc.). The criterion to optimize is often standard (makespan). Problems have been investigated and classified according to their computational complexity. A number of methods have been proposed (exact methods, greedy algorithms, approximate methods, etc.), depending on the difficulty of a particular problem. These kinds of studies are the most

common in the literature devoted to scheduling problems, and there are many books dealing with the most classic problems (see for example [BLA 01, BRU 07, PIN 01]).

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

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

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

1.3. Uncertainty management

This section summarizes the sources of uncertainty for scheduling problems and shows that all data may be concerned. The different models that take into account these uncertainties are presented and the different approaches proposed in the literature are then reviewed. Recent literature is too rich to make a complete state-of-the-art survey

possible. We restrict ourselves to some basic works, leaving the more specialized studies in the bibliographies of the different chapters.

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

1.3.1. Sources of uncertainty

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

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

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

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

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

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

1.3.2. *Uncertainty of models*

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

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

Data values are therefore often regarded as “simply” uncertain. However, it is usually possible to maintain values within some limits, in almost all cases within a set which is discrete or continuous (interval). In the case of discrete sets, we obtain a finite but potentially large number of scenarios (a value is assigned to all data). It may be possible to allocate a probability to each scenario, even if it is not exactly computable, thus indirectly achieving a stochastic model. Even in the continuous case, it is possible to proceed using these intervals. The theory of fuzzy sets is applicable here, the

application of which to scheduling problems has seen some recent developments. This is not addressed in this book, but has been the subject of a book published by Hapke and Slowinski [SLO 00]; see also Dubois *et al.* [DUB 03].

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

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

1.3.3. Possible methods for problem solving

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

1.3.3.1. Full solution process of a scheduling problem with uncertainties

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

- Step 0: defining a static problem. The definition includes, in addition to the classical specifications in deterministic scheduling, the specifications of uncertainties and their modeling. The concept of schedule quality must also be specified at this stage.
- Step 1: computing a set of solutions, i.e. a family of feasible schedules achievable by a static algorithm α (static phase). A set of solutions can be obtained from a single solution, for example when the start times of some tasks may vary within a known interval.
- Step 2: during execution, a unique solution is calculated, that is to say the schedule actually carried out, which is the outcome of applying a dynamic algorithm δ (dynamic phase) to the set of possible solutions.

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

1.3.3.2. *Proactive approach*

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

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

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

1.3.3.3. *Proactive/reactive approach*

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

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

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

1.3.3.4. *Reactive approach*

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

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

1.4. Flexibility

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

- Time, or *temporal* flexibility, i.e. regarding the starting times of operations. This flexibility can be seen as implicit in scheduling, since it allows some operations to drift over time, if conditions dictate. This is the first level of flexibility in scheduling.
- Flexibility regarding order of execution, or *sequential* flexibility. This means being able to change the order in which the operations should run on the machines, and implicitly presupposes temporal flexibility. It can be proposed during the execution of the sequence, allowing some operations to overtake others, if the conditions require it.
- Flexibility in assignments. In cases when there are multiple copies of resources, this allows a task to be executed using a resource other than that which was initially planned. This flexibility is a great help, for example when a machine becomes unavailable. It implicitly presupposes sequential flexibility and temporal flexibility.

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

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

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

1.5. Robustness

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

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

Anyway, analysis cannot restrict itself to one solution. We are mainly interested in the performance of the process previously detailed (see section 1.3.3.1) to build these

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

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

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

- \mathcal{P} : a static problem, together with uncertainties description. Hence \mathcal{P} is a set, possibly infinite, of instances of the deterministic problem.
- \mathcal{I} : an effective instance of \mathcal{P} (describing the effective conditions met during execution) also called the scenario.
- S : an effective schedule, obtained by the studied building process. S varies according to the considered scenario and in cases of possible ambiguity it is noted $S_{\mathcal{I}}$.
- $z_{\mathcal{I}}(S)$: performance of schedule S realized on \mathcal{I} , simply denoted $z_{\mathcal{I}}$ if there is no ambiguity.
- $z_{\mathcal{I}}^*$: performance of an optimal schedule on \mathcal{I} .

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

1.5.1. Flexibility as a robustness indicator

As said before, flexibility is the freedom allowed at execution phase for building the final schedule. Intuitively, it should be easier to propose a robust method if the allowed flexibility is large. Let us think about this. If everything is possible, we could

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

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

1.5.2. Schedule stability (solution robustness)

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

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

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

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

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

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

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

$$R'_2 = \max_{\mathcal{I} \in \mathcal{P}} |z_{\mathcal{I}} - z_{\tilde{\mathcal{I}}}| \quad (\text{absolute difference})$$

R_2 can be called the *stability ratio*. In one important case, only one schedule is built whatever the scenario (this implies the feasibility hypothesis). In fact, a family of schedules is usually considered, obtained from an initial schedule by accepting some amount of temporal flexibility. The robustness of this schedule S is measured by R_2 or R'_2 , $z_{\mathcal{I}}$ being here the performance of S for \mathcal{I} .

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

When statistical data are available, a stochastic model can be used and it is possible to look for schedules whose mean behavior is good. Although the robustness measures are obtained from R_2 or R'_2 by replacing the maximum by, for instance, the expectation, it is less natural to speak of stability (except that it means some guarantees can be obtained about, for instance, the mean behavior). The obtained measures are here the traditional measures in stochastic optimization, particularly:

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

The drawback of all these measures is that they sometimes lead to schedules quite far from the best solution for a given scenario, even if it is reassuring to build a stable

set of solution, or a set with good mean behavior. That is why many authors keep the term robustness for other measures.

1.5.4. Respect of a fixed performance threshold

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

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

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

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

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

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

1.5.5. Deviation measures with respect to the optimum

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

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

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

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

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

1.5.6. Sensitivity and robustness

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

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

1.6. Bibliography

- [AÏS 07] AÏSSI H., BAZGAN C. and VANDERPOOTEN D., “Min-max and min-max regret versions of some combinatorial optimization problems: a survey”, p. 1–32, Annales du Lamsade. ROY B., ALOULOU M.A. and KALAI R. (Eds.): *Robustness in OR-DA*, Paris, 2007.
- [ALO 02] ALOULOU M.A., PORTMANN M.-C. and VIGNIER A., “Predictive-reactive scheduling for the single machine problem”, *Proceedings of the 8th International Workshop on Project Management and Scheduling*, Valencia, Spain, p. 39–42, 2002.

- [ART 99] ARTIGUES C., ROUBELLAT F. and BILLAUT J.-C., “Characterization of a set of schedules in a resource-constrained multi-project scheduling problem with multiple modes”, *International Journal of Industrial Engineering*, vol. 6, no. 2, p. 112–122, 1999.
- [ART 05] ARTIGUES C., BILLAUT J.-C. and ESSWEIN C., “Maximization of solution flexibility for robust shop scheduling”, *European Journal of Operational Research*, vol. 165, no. 2, p. 314–328, 2005.
- [AVE 00] AVERBAKH I., “On the complexity of a class of combinatorial optimization problems with uncertainty”, *Mathematical Programming*, vol. Ser A 90, p. 263–272, 2000.
- [BEN 99] BEN-TAL A. and NEMIROVSKI A., “Robust solutions of uncertain linear programs”, *Operations Research Letters*, vol. 25, p. 1–13, 1999.
- [BLA 01] BLAZEWICZ J., ECKER K.H., PESCH E., SCHMIDT G. and WEGLARZ J., *Scheduling Computer and Manufacturing Processes*, Springer-Verlag, Berlin, 2nd edition, 2001.
- [BRU 07] BRUCKER P., *Scheduling Algorithms*, Springer-Verlag, Berlin, 5th edition, 2007.
- [CHR 95] CHRÉTIENNE P., COFFMAN JR. E.G., LENSTRA J.K. and LIU Z. (Eds.), *Scheduling Theory and its Applications*, John Wiley & Sons, 1995.
- [DAN 95] DANIELS R.L. and KOVELIS P., “Robust scheduling to hedge against processing time uncertainty in single stage production”, *Management Science*, vol. 41, no. 2, p. 363–376, 1995.
- [DAN 97] DANIELS R.L. and CARILLO J.E., “ β -robust scheduling for single-machine systems with uncertain processing times”, *IIE Transactions*, vol. 29, p. 977–985, 1997.
- [DAV 00] DAVENPORT A.J. and BECK J.C., A survey of techniques for scheduling with uncertainty, available in <http://www.eil.utoronto.ca/profiles/chris/chris.papers.html>, 2000.
- [DUB 03] DUBOIS D., FARGIER H. and FORTEMPS B., “Fuzzy scheduling: modelling flexible constraints vs coping with incomplete knowledge”, *European Journal of Operational Research*, vol. 147, p. 231–252, 2003.
- [FIA 98] FIAT A. and WOEGINGER G.J. (Eds.), *Online Algorithms, The State of the Art*, Lecture Notes in Computer Science, Springer, 1998.
- [GOL 97] GOLDRATT E.M., *Critical Chain*, The North River Press Publishing Corporation, Great Barrington, 1997.
- [GRA 79] GRAHAM R.E., LAWLER E.L., LENSTRA J.K. and RINNOY KAN A., “Optimization and approximation in deterministic sequencing and scheduling: a survey”, *Ann. Discrete Math.*, vol. 4, p. 287–326, 1979.
- [HAL 04] HALL N. and POSNER M., “Sensitivity analysis for scheduling problems”, *Journal of Scheduling*, vol. 7, p. 49–83, 2004.

- [HER 01] HERROELEN W. and LEUS R., “On the merits and pitfalls of critical chain scheduling”, *Journal of Operations Management*, vol. 19, no. 5, p. 559–577, 2001.
- [HER 05] HERROELEN W. and LEUS R., “Project scheduling under uncertainty, survey and research potential”, *European Journal of Operational Research*, vol. 165, p. 289–306, 2005.
- [KOU 97] KOUVELIS P. and YU G., *Robust Discrete Optimisation and its Applications*, Kluwer Academic Publishers, 1997.
- [KOU 00] KOUVELIS P., DANIELS R.L. and VAIRAKTARAKIS G., “Robust scheduling of a two-machine flow shop with uncertain processing times”, *IIE Transactions*, vol. 32, p. 421–432, 2000.
- [MOU 99] MOUKRIM A., SANLAVILLE E. and GUINAND R., “Scheduling with Communication Delays and On-line Disturbances”, in AMESTOY P. (Ed.), *Euro-Par'99, Toulouse, France*, LNCS 1685, p. 350–357, 1999.
- [MUL 95] MULVEY J.M., VANDERBEI R.J. and ZENIOS S.A., “Robust optimization of large scale systems”, *Operations Research*, vol. 43, p. 264–281, 1995.
- [PEN 01] PENZ B., RAPINE C. and TRYSTRAM D., “Sensitivity analysis of scheduling algorithms”, *European Journal of Operational Research*, vol. 134, p. 606–615, 2001.
- [PIN 01] PINEDO M., *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, Englewood Cliffs, 2nd edition, 2001.
- [ROY 02] ROY B., “Robustesse de quoi et vis-à-vis de quoi mais aussi robustesse pourquoi en aide à la décision?”, *Newsletter of the European Working group “Multicriteria Aid for Decisions”*, vol. 3, no. 6, p. 1–6, 2002.
- [SGA 98] SGALL J., “On-line scheduling – a survey”, p. 196–231, in Fiat and Woeginger (Eds.) [FIA 98].
- [SLO 00] SLOWINSKI R. and HAPKE M., *Scheduling Under Fuzziness*, Physica-Verlag, Heidelberg, 2000.
- [SOT 98] SOTSKOV Y.N., WAGELMANS A. and WERNER F., “On the calculation of the stability radius of an optimal or an approximate schedule”, *Annals of Operational Research*, vol. 83, p. 213–225, 1998.
- [T'K 06] T'KINDT V. and BILLAUT J.-C., *Multicriteria scheduling*, Springer, Berlin, 2nd edition, 2006.
- [WEI 95] WEISS G., “A tutorial in stochastic scheduling”, p. 33–64, in Chrétienne *et al.* [CHR 95].
- [WU 99] WU S.D., BYEON E.S. and STORER R.H., “A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness”, *Operations Research*, vol. 47, no. 1, p. 113–124, 1999.

Chapter 2

Robustness in Operations Research and Decision Aiding

It is always advisable to perceive clearly our ignorance.
(Charles Darwin)

2.1. Overview

The search for robustness is an ever present concern in Operations Research and Decision Aiding (OR-DA) where increasingly rich and diversified methodologies and concepts are emerging. The term “robust” applies to a large variety of objects such as solution, method and conclusion. In OR-DA, the notion of robustness is often used in the same way as (sometimes even instead of) flexibility, stability, sensitivity and even equity in certain cases.

Faced with this diversity, I think it is necessary to highlight what seems to be the generally used meaning for the term “robust” in OR-DA (section 2.1.1) before specifying the reasons leading to the concern for robustness in this discipline (section 2.1.2) and presenting the structure of this chapter (section 2.1.3).

Chapter written by Bernard ROY.

2.1.1. Robust in OR-DA with meaning?

As I see it, *robust* is a term that is generally used in the sense of *a capacity for withstanding “vague approximations” and/or “zones of ignorance” in order to prevent undesirable impacts, notably the degradation of the properties to be maintained.*

“Vague approximations” can refer to a way of modeling, the restrictive character of certain hypotheses, mode of value allocation to data and/or parameters, etc.

“Zones of ignorance” may deal with the complexity of certain phenomena and of value systems but mostly the future: trends, contingencies, behavior of others, etc.

Here are a few examples to illustrate this meaning of the term “robust” in scheduling. In Chapter 1, a solution is said to be robust if its “performance is rather insensitive to data uncertainties and disturbances”. In this context, insensitivity to data uncertainty means resisting to this uncertainty¹. The uncertainty in question can for example refer to the way of modeling which processes certain data as insignificant or not influenced by contingencies, a Gaussian hypothesis simplifying the mode of consideration of a random phenomenon or the approximate character of values attributed to data (processing time, due date, etc.).

In some maintenance studies, scheduling must be conceived to guarantee deadlines are respected even though the jobs to be done are not well known (resistance to a certain form of ignorance). Job-shop scheduling may have to be chosen for its capacity to face an order book that is only partially known or with unknown reactions to delays that the end customer may encounter because of contingencies. Climate conditions, as well as work-related accidents or social upheavals, are sources of ignorance that project management may have to consider.

Two comments seem necessary to specify the meaning of what was just discussed:

- 1) Even though the borderline between *vague approximations* and *zones of ignorance* is far from being well defined, all vague approximations do not come from a zone of ignorance and all zones of ignorance do not lead to vague approximations.

1. The term “uncertainty” imperfectly covers all forms of vague approximations and zones of ignorance that need to be resisted. This is the case in particular of vague approximations resulting from simplifications or ill determinations. This is also the case for zones of ignorance coming from certain forms of imperfect knowledge relative to the complexity of phenomena or value systems.

2) Resistance can have the following meanings: protecting from, adapting to, being rather insensitive to, remaining stable, settling a certain form of equity, etc.

We will now examine the reasons resulting in the need to resist these vague approximations and zones of ignorance in OR-DA.

2.1.2. Why the concern for robustness?

In OR-DA, the capacity for resistance qualified by robustness is required in order to be protected from undesirable impacts, impacts that should be apprehended taking into account these vague approximations and/or zones of ignorance that need to be resisted. The nature of these impacts, along with the (very often subjective) way of assessing their undesirable character, are contingent to the context involved. The concerns motivating the search for robustness are extremely diversified for these reasons. I will settle for illustrating them through a series of examples in this chapter:

i) Exceptional character decisions

- Layout of a large linear infrastructure (very high speed train line, highway, high-tension line, etc.): throughout the execution (five years or more), what reactions will it generate? Once this is finished, what standards will it be judged by? Will the size be adapted to traffic?

- Construction of a sanitation or waterworks system: knowing that implementing such activities, as with the evolution of consumption patterns, can only be defined in large variation ranges, will the designed system be able to fulfill population requirements in the planned horizon without needing adjustments leading to prohibitive costs?

- Updating of equipment: considering the evolution of technology and environmental standards, when should the decision be made to update?

ii) Sequential character decisions

- Plan designed to be implemented in stages: how will the contexts in future stages be affected by the decisions taken at this present stage? Do they allow for possible evolutions of these contexts by keeping the range of adaptations and reactions open?

- Scheduling flight personnel in an airline company: how can we handle unexpected unavailability of teams (unforeseen absences, immobilization during a mission, etc.) in acceptable economic conditions and with no planning disruptions for agents?

iii) Choice of a method for repetitive applications

– Management support method for restocking a store: does the method protect against out of stock risks that could result from a failure to respect of delivery lead times by suppliers? Is it adapted for possible evolutions of purchase agreements?

– Method controlling budget distribution between members in a group: knowing that the size and composition of beneficiary groups can greatly change over time and space, will the method retained be considered fair in all cases where it will be applied?

– Adjustment method for a model dedicated to emphasizing the way in which different factors contribute to global client satisfaction during consecutive surveys: how can we avoid the results depending on final retained values (chosen in a relatively arbitrary manner in certain intervals) for different technical parameters involved in the model?

2.1.3. Plan of the chapter

In the next section, I will examine where, for a decision aiding problem (DAP), “vague approximations” and “zones of ignorance” come from, for which the need for protection leads to the search for robustness. These vague approximations and zones of ignorance are closely linked to the way that *the decision aiding problem is formulated* (DAPF). They can also depend (although generally less so) on the processing procedure applied to this formulation in the decision aiding process. This leads me to introduce the general concept of *version*. In section 2.3, I will specify the meaning I give to several currently used terms (procedure and method notably) in order to clarify their links with the concern for robustness. In section 2.4, I will focus on the way to take robustness into consideration: what must be robust? How can we formalize robustness? In what form can vague approximations and zones of ignorance be taken into account? Unfortunately, many questions raised here will remain unanswered. A brief conclusion will complete this chapter.

2.2. Where do “vague approximations” and “zones of ignorance” come from? – the concept of version

2.2.1. Sources of inaccurate determination, uncertainty and imprecision

Once the DAP is formulated, we should identify, in this formulation, the “places” (concepts, numerical data, presence or absence of links between phenomena, neglected aspects and factors, way to formulate constraints and criteria, the arbitrariness of certain operating instructions in a process mode, etc.) which could be affected by vague approximations and/or zones of ignorance. The specific form in which these relevant places are presented is obviously very specific to the problem

studied, the way it is formulated and the process mode applied. Nevertheless, I think it is possible to see them whatever they are as *frailty points* connected to *sources of inaccurate determination, uncertainty or arbitrariness* (see [ROY 89]). The vague approximations and zones of ignorance that must be resisted actually come from such sources. They can, it seems, be classified into three categories (even though the line separating these categories is not perfectly well defined, they affect sides of the DAPF which I think it is important to distinguish):

- Source $S \cdot \alpha$: vague, uncertain, unknown, and even undetermined character of factual data, objective descriptions of phenomena and purely technical procedural aspects in relation to the form in which they must occur during the aiding process in the present situation.

This source may, for example, affect frailty points such as: processing times, due dates, process cost, failure probabilities, probability distributions chosen for modeling a random factor, discrimination thresholds, values given to the parameters playing a mostly technical role in a model or procedure, techniques used to adjust a model intended to represent complex phenomena, etc.

- Source $S \cdot \beta$: implementation conditions of the decision that must be taken; these conditions can be influenced by the future state of the environment:

- during implementation if the decision is punctual (i.e., taken all at once);
- by consecutive environmental steps if the decision is sequential.

This source may, for example, affect frailty points such as: what will have happened (during implementation), labor and/or raw material cost, interest rates, consumption patterns, boundaries of what is acceptable (social and environmental standards among others) or the presence or absence of disrupting events (unavailable personnel or equipment, opposition of some stakeholders, climatic incidents, etc.).

- Source $S \cdot \gamma$: eminently subjective character of different aspects (not dealing with sources $S \cdot \alpha$ and $S \cdot \beta$) dealing with feasibility, relative interest and process modes of the different potential actions, especially the fuzzy, unstable and possibly incoherent and/or incomplete character of value systems which are supposed to prevail in the decision aiding process.

This source may, for example, affect frailty points such as the role devoted to certain criteria (notably on the basis of values allocated to substitution rates, weight, veto thresholds, etc.), the level required to validate a majority or set a cut-off threshold, the mode of appreciation for limits marking the feasibility or boundary between categories, the way to code a qualitative dimension by means of an interval

scale, the way to apprehend attitude toward risk, the place reserved for certain actors (notably future generations).

2.2.2. DAP formulation: the concept of version

The expression *DAP formulation* must be taken in a very general sense. It obviously includes the model, insofar as there is modeling (see section 2.3.3 below), but in a broader sense, everything that was in question and has finally been retained to contain and consequently *formulate the problem*, including the problematic (the way in which decision aiding was conceived, see [ROY 96], Chapter 6), the properties to preserve and, in general, the undesirable impacts from which we want to be protected.

When we start being concerned about robustness in OR-DA, it is necessary in my opinion to start by identifying in DAPF what I have called frailty points connected to each of the three types of sources $S \cdot \alpha$, $S \cdot \beta$, $S \cdot \gamma$. Relative to each of these frailty points, we should then explain the different *options* which deserve to be considered within this formulation in order to take inaccurate determination, uncertainty and arbitrariness margins into consideration from these sources. The selection of a specific option for each of the identified frailty points defines what I proposed (see [ROY 02, ROY 07]) to call a *version* of DAPF. If it is carried out without precautions, this selection can very well lead to combinations lacking in coherence or plausibility. Let \hat{V} be the set of versions V corresponding to combinations of options deemed (possibly from very subjective bases) to be of interest (\hat{V} cannot be discrete). From this definition, two versions of \hat{V} can notably differ by:

1) The values assigned to certain factual data or technical parameters characterizing events, phenomena, etc.: this is the case in particular with vague approximations and zones of ignorance from $S.\alpha$; when in the DAPF, this source is the most significant, the word *version* becomes synonymous with *instance* or *datasets*.

2) The way that we describe the future universe in which the decision must be executed: this is the case in particular with vague approximations and zones of ignorance from $S.\beta$; when in the DAPF, this source is the most significant, the word *version* becomes synonymous with *scenario*.

3) The way in which ambiguities, uncertainties and the multiplicity of value systems are taken into consideration: with vague approximations and zones of ignorance from $S.\gamma$ this is particularly the case for; when in the DAPF, this source is the most significant, the word *version* becomes synonymous with *interpretation* or *mode of appreciation*.

I think it is useful to emphasize that the way in which \hat{V} versions distinguish themselves does not generally come from only one of the three sources. The search for robustness must rely on what comes from each source in order to arrive at an appropriate design (see section 2.4.3) of the set \hat{V} . However, the processing procedure of these versions in a perspective of decision aiding can also be *affected* (in certain cases, we could also say “infected”) by sources $S \cdot \alpha, S \cdot \beta, S \cdot \gamma$ as will be emphasized in the next section.

2.3. Defining some currently used terms

These refinements seem useful to clarify what is commonly applied to the term “robust” in OR-DA as well as the way in which the search for robustness can be guided. With this goal in mind, I will first explain what I mean when I use the terms *procedures*, *results* and *methods*. I will then discuss the existence of two types of procedures and methods commonly used in OR-DA. Finally, since the search for robustness generally leads to emphasizing a certain number of results in order to reach conclusions, I will explain how I use this last term.

2.3.1. Procedures, results and methods

A *procedure* P represents a set of execution instructions for handling a problem that will produce a $R(P, V)$ result when applied to a version V of a DAPF. These execution instructions often present frailty points (I will give a few examples during the presentation of a method) which always have $S \cdot \alpha, S \cdot \beta, S \cdot \gamma$ as sources.

Result is used to refer to the outcome of applying P to a rigorously formulated problem. A $R(P, V)$ result can have different forms, with the main ones as follows:

- 1) Solutions² or bundles possessing the required properties: admissible, not dominated according to a family of criteria, optimal according to a criterion, etc.;
- 2) Statements: absence of solutions with such property(ies)..., observed inconsistencies or incompatibilities are as follows ... and they have this origin..., that solution has this performance and its gap in relation to the optimum equals..., this solution is not dominated, etc.

For different reasons, the search for robustness can lead to the application of several approaches to the same decision aiding process: sources $S \cdot \alpha, S \cdot \beta, S \cdot \gamma$ can for example justify the involvement of a set \hat{P} of procedures.

2. In the context studied, the way that a problem is formulated is what leads us to agree on the meaning that we give to the term “solutions”.

A *method M* here designates³ a family \hat{P} of *similar* procedures, i.e., they satisfy both the following requirements:

- 1) \hat{P} procedures act as members of the same class because of their common features (structure, concepts, axioms or hypotheses, etc.).
- 2) Procedure class members are only different by the options taken in relation to certain frailty points of the method (examples: concordance level or cut-off threshold in ELECTRE methods, thresholds used to make strict certain inequalities in MACBETH or MUSA, multiple parameters occurring in Tabu search, simulated annealing, genetic algorithm methods⁴, etc.) and/or by different consideration modes (not necessarily formalized) of certain aspects of the version to which they are applied (examples: a way to exploit certain liberties offered in scheduling methods, see Chapter 1, role of a function as criterion or constraint in multiobjective programming⁵, expert subjectivity in an expertise type method (see section 2.3.2 below), etc.).

2.3.2. Two types of procedures and methods

In OR-DA, we use two types of procedures and consequently two types of methods.

Algorithmic procedures (AP): these are procedures in which the execution instructions for the processing procedure are formalized enough to be trusted to a “machine” with no human intervention. In order to be applied to a version, this type of procedure requires that this version has also received a complete and rigorous formal definition.

By *algorithmic methods* (AM), I designate a method where all procedures are AP.

Expertise procedures (EP): these are procedures in which execution instructions for the processing procedure require the intervention of a human, here called the expert. With this term, I particularly include the one that has to intervene as decision maker in an interactive procedure⁶. Procedures of this type can be applied to versions which have not received a complete and rigorous formal definition. Because the result

3. In relation with the terminology of [VIN 99a, VIN 99b]; however, Vincke did not impose the further restrictive conditions.

4. For more information on these methods, see [SOR 01, GRI 02, BAN 05] and [FIG 05b].

5. For more information on these methods, see [EHR 02].

6. For example, see [ROY 93, BEU 01, DIA 02, DIA 07, GRE 07, AÏT 04, BAN 05, KOR 05, SAA 05, FIG 08].

can depend on the expert, each EP procedure must incorporate the personality of the expert in its definition (contrary to the machine which normally does not get involved in an AP procedure definition). EP procedures can obviously include algorithmic phases.

By *expertise methods* (EM), I designate a method where all procedures are EP. In this type of method, I include, for example, those involving several experts in a single DAPF⁷, the expert being for instance different from one procedure to the next.

A procedure and, more importantly, a method applied to several versions of a DAPF produce a set of results that robustness research must use.

2.3.3. Conclusions relative to a set \hat{R} of results

A set \hat{V} of versions and a set \hat{P} of procedures that can be applied to these versions being given, I propose calling (see [ROY 97, ROY 98]) *conclusion* any assertion (deemed valid) which exploits all or a part of a set \hat{R} of results obtained from (P, V) pairs, elements of a set $\hat{T} \subset \hat{P} \times \hat{V}$. Considering the previous definitions, there follows a few assertion examples which may constitute conclusions:

- Except for the following procedures... all $(P, V) \in \hat{T}$ pairs reveal the following solutions... as admissible with the following evaluations...
- For all pairs of \hat{T} , S is a solution for which the gap to optimum never exceeds a given threshold.
- Results from \hat{R} obtained from a sampling $T \subset \hat{T}$ reveal the following invariants...
- The notable admissible solutions present in \hat{R} lead to bring to light the following variation margins in relation to the performance of considered criteria...
- Except for a few pairs (P, V) , a given set S of solutions benefits from the following properties...
- The following objectives... are irreconcilable when we only consider the pairs of a given subset $T \subset \hat{T}$.

2.4. How to take the robustness concern into consideration

By using the previous definitions and conventions, it is now possible to specify (see section 2.4.1) how the term “robust” is usually applied in OR-DA. This will then

7. See, for example, [SAL 95, ROU 96, KIM 98, PIC 01, DAM 02].

lead me to raise some questions (without claiming to have many answers) concerning validating robustness (see section 2.4.2) and conceiving the sets \hat{P} , \hat{V} , \hat{T} (see section 2.4.3).

2.4.1. What must be robust?

I would first like to shed light on the fact that, regardless of what is qualified as “robust”, robustness involved is relative to the set of pairs (P, V) considered. Second, I would like to emphasize the fact that this term “robust” is not relevant for qualifying the $R(P, V)$ product resulting from the application of a single procedure P to a single version V . On the other hand, it can qualify:

1) A (AP or EP) *procedure P* by reference to a set \hat{V} of versions in the case where, for each of these versions, P provides a solution, or a set of solutions, or a statement considered adapted to the version involved, in relation to one or more criteria to define it. Several chapters of this book examine, explicitly or otherwise, the robustness of a procedure, often called an algorithm (for example see Chapters 3, 4, 6 and 9).

2) A *solution S* by reference to a set \hat{T} not reduced to a single pair (P, V) : for example, we will say that S is robust because it is always admissible or it is always at most $\epsilon\%$ from the optimum in all cases studied; other example: value λ_0 of parameter λ occurring in a set \hat{P} of adjustment procedures constitutes a robust solution because, regardless of the versions $V \in \hat{V}$ considered, the model correctly reports observations.

3) A set \mathbf{S} of solutions: this can be qualified as robust, for example:

- in the case where it is built from a set \hat{P} of procedures applied to a single version V if these solutions are compatible or do not reveal a contradiction in a previously defined sense;

- in the case where it is built applying a single procedure to a set \hat{V} of versions if, $\forall V \in \hat{V}$, in S there exists at least one solution considered satisfactory, in a well defined sense, for this version (see section 1.3.3.3).

4) A *conclusion* similar to those illustrated in section 2.3.3 above: qualifying this robustness conclusion means that its validity is recognized in conditions which deserve to be clearly made explicit. Before clarifying this point, I would like to point out that the assertion of robustness of a procedure, a solution or a set of solutions (as it was considered previously) is none other than a relatively familiar form of robust conclusions. I have shown (see [ROY 07, ROY 98]) the usefulness of distinguishing at least three types of validation conditions for a conclusion qualified as robust. They are briefly described below:

- A conclusion is qualified as *perfectly robust* when it is rigorously validated for all pairs of a well defined set $\hat{T} \subset \hat{P} \times \hat{V}$.
- A conclusion is qualified as *approximately robust* when it is rigorously validated for “almost” all pairs (P, V) of a set $\hat{T} \subset \hat{P} \times \hat{V}$, “almost” meaning that exceptions are relative to pairs (P, V) which are not necessarily well identified but considered insignificant in the sense that they involve combinations $P \times V$ not considered interesting or relevant.
- A conclusion is qualified as *pseudo-robust* when it makes a statement that is not necessarily rigorously formalized but considered valid for most pairs (P, V) of a set \hat{T} ; the validity judgment can for instance rely on the results obtained for a sampling T of \hat{T} pairs.

5) A *method M*: defining the conditions that must be fulfilled to qualify a method as robust can be very different according to contexts (in particular, see Chapter 1 and [VIN 99a, VIN 99b]; [SOR 01]; [SLO 03]). These definitions can in particular be different whether the method must be applied to a single well identified version or to a set of versions. I now propose to qualify a method as robust in relation to a version or a set of versions to which it is applied:

- for *AM* if solution S or set \mathbf{S} of solutions produced is robust in a well defined sense (see 1 and 2 above);
- for *EM* if it is possible to arrive at validated conclusions (see section 2.3 above) by the expert(s) involved.

These examples shed light on the fact that in all cases, the term “robust” must relate to set \hat{T} or subset $T \subset \hat{T}$ of pairs (P, V) considered for validating the type of robustness desired.

2.4.2. What are the conditions for validating robustness?

The considerations in the previous section highlight the variety to which the term “robust” can be applied to as well as the diversity of validation conditions which can be used to accept or reject robustness. Considering the DAPP and what we want to apply the term “robust” to, choosing these validation conditions greatly depends on the nature of undesirable impacts from which we want to be protected as well as the way in which we must be protected. In order to set these validation conditions, we can attempt to answer questions such as:

- a) Will the validation conditions operate by *acceptance-rejection* (admissibility, respect for a performance threshold, etc.) or by using a *degree of robustness* (see b

below) or still by being *qualified* in different ways (perfectly robust, approximately robust, pseudo-robust, etc.)?

b) When the concept of solution is present⁸ in what the term “robust” must apply to, must undesirable impacts be understood in terms of:

- *efficiency*: performance level, cost difference (relative or absolute) in relation to an optimum, etc. (see for example Chapters 4, 5 and [SEN 91, RIO 94, ESC 94, KOU 97, VAL 99, HIT 00, JEU 00, AIS 07, KAL 07, ROY 07]);

- *flexibility*: possibilities of adaptation, openness to the future, etc. (see for example Chapters 9, 10, 11 and [GUP 72, ALO 01, ROS 01a, ROS 01b, ROS 01c, SEV 02, ALO 07, SOR 07]);

- *stability*: performance gap between solutions relative to the different version pairs or between a reference solution or relative to the different versions (see for example Chapters 1, 3, 12 and [SAN 07]);

- *equity*: balance of a certain distribution (see for example [PER 03, SPA 03])?

c) Is it necessary to be able to compare the robustness of solutions, of set of solutions, of conclusions or methods: if that is the case, must we define a single criterion or a set of criteria?

d) Must the results relative to those of pairs $(P, V) \in \hat{V}$ which, in some respects, seem to be the worst, play a decisive role in the way to understand robustness (see Chapters 4, 6 and 11; [KOU 97])? Since the worst can very well be unlikely, is a more nuanced reasoning possible by combining risk and efficiency (see [SOR 01, HIT 02])?

2.4.3. How can we define the set of pairs of procedures and versions to take into account?

The answer to this question is (except in rare cases) very subjective. Considering the nature of impacts from which the goal of robustness is to protect as well as the way we must be protected, it is useful to begin by considering the following questions:

1) How can we state the definition of a set \hat{V} of representative versions? Should they be completely formalized?

2) How can we choose a set \hat{P} of appropriate procedures? Must these procedures be of *AP* or *EP* type?

8. Examples in section 2.4.1 illustrate that this is very often the case, even though “robust” is used to qualify conclusions or a method.

3) Can set \hat{T} of pairs (P, V) to consider be restrictive or not (for incompatibility, lack of interest or processing time reasons) to a subset of $\hat{P} \times \hat{V}$?

I think it would be useful at this stage to emphasize the fact that research that is too systematic for frailty points to be affected by vague approximations and/or zones of ignorance may lead to excessive proliferation of versions and/or procedures to consider. On the other hand, an insufficient critical attitude, ignoring the saying “*a man who doesn't know that he doesn't know thinks he knows*”, can lead to reducing this proliferation excessively. In other words, it is advisable to find a compromise in each situation between these two opposite tendencies taking into account expectations of those in whose name decision aiding is occurring.

2.5. Conclusion

Those responsible for making decisions or more generally for influencing a decision making process do not expect the decision aiding to dictate their choices. They are looking for useful information that will help to restrict the scope of their deliberations and actions.

Whether this information is presented in the form of recommended solutions, suggested method or prescriptions based on conclusions, they will only really be useful if the manner in which they are dependent or conditioned by contingency, arbitrariness and ignorance liable to come from sources $S.\alpha$, $S.\beta$, $S.\gamma$ was taken into account in a sufficiently large and explicit framework.

It is therefore important that this information exploits not only a single special result $R(P, V)$ but all those associated with a set \hat{T} of pairs (P, V) built from sets \hat{P} and \hat{V} . These sets must actually be conceived to respond to this robustness concern.

2.6. Bibliography

- [AIS 07] AISSI H., BAZGAN C. and VANDERPOOTEN D., “Min-max and min-max regret versions of some combinatorial optimization problems: a survey”, p. 1-32, in ROY B., ALOULOU M.A. and KALAÏ R. (Eds.), *Robustness in OR-DA*, Annales du LAMSADE no. 7, Paris-Dauphine University, 2007.
- [AÏT 04] AÏT YOUNES A. and ROY B., “Prise en compte d'une connaissance imparfaite à l'aide d'un pseudo-critère: Procédure interactive de construction”, *Journal of Decision Systems*, vol. 13, no. 1, p. 113–142, 2004.

- [ALO 01] ALOULOU M.A. and PORTMANN M.-C., “Définition d’ordonnancements flexibles. Première application à un problème à une machine”, *Conférence de Génie Industriel GI'2001*, Aix-en-Provence, p. 1029–1038, 2001.
- [ALO 07] ALOULOU M.A. and ARTIGUES C., “Flexible solutions in disjunctive scheduling: general formulation and study of the flow-shop case”, p. 33–51, in ROY B., ALOULOU M.A. and KALAI R. (Eds.), *Robustness in OR-DA*, Annales du LAMSADE no. 7, Université Paris-Dauphine, 2007.
- [BAN 05] BANA E COSTA C., VANSNICK J.C. and DE CORTE J.M., “*MACBETH*”, in Figueira *et al.* [FIG 05a], 2005.
- [BEU 01] BEUTHE M. and SCANNELLA G., “Comparative analysis of UTA multicriteria methods”, *European Journal of Operational Research*, vol. 130, no. 2, p. 246–262, 2001.
- [DAM 02] DAMART S., MOUSSEAU V. and SOMMERLATT I., “Du mode d’implication d’acteurs multiples dans le cadre de l’utilisation d’un modèle d’affectation multicritère: analyse au regard d’une application à la tarification des transports publics”, *INFOR*, vol. 40, no. 3, p. 199–222, 2002.
- [DIA 02] DIAS L., MOUSSEAU V., FIGUEIRA J. and CLIMACO J.N., “An aggregation/disaggregation approach to obtain robust conclusions with ELECTRE TRI”, *European Journal of Operational Research*, vol. 138, no. 2, p. 332–348, 2002.
- [DIA 07] DIAS L.C., “A note on the role of robustness analysis in decision-aiding processes”, p. 53–70, in ROY B., ALOULOU M.A. and KALAI R. (Eds.), *Robustness in OR-DA*, Annales du LAMSADE no. 7, Paris-Dauphine University, 2007.
- [EHR 02] EHRGOTT M. and GANDIBLEUX X. (Eds.), *Multiple Criteria Optimization. State of the Art Annotated Bibliographic Surveys*, Kluwer Academic Publishers, Dordrecht, 2002.
- [ESC 94] ESCUDERO L.F., “Robust decision making as a decision making aid under uncertainty”, p. 126–138, in Rios [RIO 94].
- [FIG 05a] FIGUEIRA J., GRECO S. and EHRGOTT M. (Eds.), *Multiple Criteria Decision Analysis: The State of the Art Surveys*, International Series in Operations Research and Management Sciences, Kluwer Academic Publishers, 2005.
- [FIG 05b] FIGUEIRA J., MOUSSEAU V. and ROY B., “ELECTRE methods”, in Figueira *et al.* [FIG 05a].
- [FIG 08] FIGUEIRA J., GRECO S. and SLOWINSKI R., “Building a set of additive value functions representing a reference preorder and intensities of preference: GRIP method”, *European Journal of Operational Research*, 2008.
- [GRE 07] GRECO S., MOUSSEAU V. and SLOWINSKI R., “Robust multiple criteria ranking using a set of additive value functions”, p. 95–128, in ROY B., ALOULOU M.A. and KALAI R. (Eds.), *Robustness in OR-DA*, Annales du LAMSADE no. 7, Paris-Dauphine University, 2007.

- [GRI 02] GRIGOURIDIS E. and SISKOS Y., "Preference disaggregation for measuring and analysing customer satisfaction: the MUSA method", *European Journal of Operational Research*, vol. 143, no. 1, p. 148–170, 2002.
- [GUP 72] GUPTA S.K. and ROSENHEAD J., "Robustness in sequential investment decisions", *Management Science*, vol. 15, no. 2, p. 18–29, 1972.
- [HIT 00] HITES R., The robust shortest path problem, PhD Thesis, Université Libre de Bruxelles, Brussels, Belgium, 2000.
- [HIT 02] HITES R., The aggregation of preferences method for solving certain combinatorial problems with uncertainty, Report no. preprint, <http://smg.ulb.ac.be>, 2002.
- [JEU 00] JEUNET J. and JONARD N., "Measuring the performance of lot-sizing techniques in uncertain environments", *International Journal of Production Economics*, vol. 64, p. 197–208, 2000.
- [KAL 07] KALAI R. and LAMBORAY C., "L' α -robustesse lexicographique : une relaxation de la β -robustesse", p. 129–143, in ROY B., ALOULOU M.A. and KALAI R. (Eds.), *Robustness in OR-DA Annales du LAMSADe* no. 7, Paris-Dauphine University, 2007.
- [KIM 98] KIM S.H., CHOI S.H. and AHN B.S., "Interactive group decision process with evolutionary database", *Decision Support System*, vol. 23, p. 333–345, 1998.
- [KOR 05] KORHONEN P., "Interactive Methods", in Figueira *et al.* [FIG 05a].
- [KOU 97] KOUVELIS P. and YU G., *Robust Discrete Optimization and its Applications*, Kluwer Academic Publishers, Dordrecht, 1997.
- [PER 03] PERNY P. and SPANJAARD O., "An axiomatic approach to robustness in search problems with multiple scenario", *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence*, Acapulco, Mexico, p. 469–476, 2003.
- [PIC 01] PICTET J. and BELTON V., "ACIDE: Analyse de la compensation et de l'incomparabilité dans la décision-Vers une prise en compte pratique dans MAVT", *EUR 19808 EN report*, COLORNI A., PARUCCINI M. and ROY B. (Eds.), *A-MCDA - Aide Multi Critère à la Décision (Multiple Criteria Decision Aiding)*, European Commission, Joint Research Center, p. 245–256, 2001.
- [RIO 94] RIOS S. (Ed.), *Decision Theory and Decision Analysis: Trends and Challenges*, Kluwer Academic Publishers, 1994.
- [ROS 01a] ROSENHEAD J., "Robustness analysis: keeping your options open", p. 181–207, in Rosenhead and Mingers [ROS 01c].
- [ROS 01b] ROSENHEAD J., "Robustness to the first degree", p. 209–222, in Rosenhead and Mingers [ROS 01c], 2001.
- [ROS 01c] ROSENHEAD J. and MINGERS J. (Eds.), *Rational Analysis for a Problematic World Revisited: Problem Structuring Methods for Complexity, Uncertainty and Conflicts*, John Wiley & Sons, Chichester, 2001.

- [ROU 96] ROUSSEAU A. and MARTEL J.-M., La décision participative: une démarche pour gérer efficacement les problèmes environnementaux, Report no. 96-24, Université Laval, Québec, Canada, 1996.
- [ROY 89] ROY B., “Main sources of inaccurate determination, uncertainty and imprecision in decision models”, *Mathematical and Computer Modelling*, vol. 2, no. 10/11, p. 1245–1254, 1989.
- [ROY 93] ROY B. and BOUYSOU D., *Aide multicritère à la décision: Méthodes et cas*, Economica, Paris, 1993.
- [ROY 96] ROY B., *Multicriteria Methodology for Decision Aiding*, Kluwer Academic Publishers, Dordrecht, 1996.
- [ROY 97] ROY B., Une lacune en RO-AD: les conclusions robustes, Report no. 144, LAMSADE notebook, Paris-Dauphine University, 1997.
- [ROY 98] ROY B., “A missing link in OR-DA, robustness analysis”, *Foundations of Computer and Decision Sciences*, vol. 23, no. 3, p. 141–160, 1998.
- [ROY 02] ROY B., “Robustesse de quoi et vis-à-vis de quoi mais aussi robustesse pourquoi en aide à la décision ?”, *Proceedings of the 56th Meeting of the European Working Group “Multiple Criteria Decision Aiding”*, FIGUEIRA J., HENGGELE-R-ANTUNES C. and CLÍMACO J. (Eds.), Coimbra, Portugal, 2002.
- [ROY 07] ROY B., La robustesse en recherche opérationnelle et aide à la décision : une préoccupation multi-facettes, in ROY B., ALOULOU M.A. and KALAI R. (Eds.), *Robustness in OR-DA*, Annales du LAMSADE no. 7, Paris-Dauphine University, 2007.
- [SAA 05] SAATY T., “The analytic hierarchy and analytic network processes”, in Figueira et al. [FIG 05a].
- [SAL 95] SALO A., “Interactive decision aiding for group decision support”, *European Journal of Operational Research*, vol. 84, p. 134–149, 1995.
- [SAN 07] SANLAVILLE E., “Sensitivity and robustness analyses in scheduling: the two machine with communication case”, p. 253–268, in ROY B., ALOULOU M.A. and KALAI R. (Eds.), *Robustness in OR-DA*, Annales du LAMSADE no. 7, Paris-Dauphine University, 2007.
- [SEN 91] SENGUPTA J.K., “Robust decisions in economic models”, *Computers and Operations Research*, vol. 18, no. 2, p. 221–232, 1991.
- [SEV 02] SEVAUX M. and SORENSEN K., “Genetic algorithm for robust schedules”, *Proceedings of the 8th International Workshop on Project Management and Scheduling (PMS'2002)*, Valencia, Spain, p. 330–333, 2002.

- [SLO 03] SLOWINSKI R., GRECO S. and MOUSSEAU V., “Assessing a partial preorder of alternatives using ordinal regression and additive utility functions-A new UTA method”, *58th Meeting of the European Working Group “Multiple Criteria Decision Aiding”*, Moscow, Russia, 2003.
- [SOR 01] SORENSEN K., “Tabu searching for robust solutions”, *Proceedings of the 4th Metaheuristics International Conference*, Porto, Portugal, 2001.
- [SOR 07] SORENSEN K. and SEVAUX M., “A practical approach for robust and flexible vehicle routing using metaheuristic and Monte Carlo sampling”, p. 269–285, in ROY B., ALOULOU M.A. and KALAÏ R. (Eds.), *Robustness in OR-DA*, Annales du LAMSADE no. 7, Paris-Dauphine University, 2007.
- [SPA 03] SPANJAARD O., Exploitation de préférences non-classiques dans les problèmes combinatoires: modèles et algorithmes pour les graphes, PhD Thesis, Paris-Dauphine University, 2003.
- [VAL 99] VALLIN P., “Détermination d'une période économique robuste dans le cadre du modèle de Wilson”, *RAIRO Recherche Opérationnelle*, vol. 33, no. 1, p. 47–67, 1999.
- [VIN 99a] VINCKE P., “Robust and neutral methods for aggregating preferences into an outranking relation”, *European Journal of Operational Research*, vol. 112, no. 2, p. 405–412, 1999.
- [VIN 99b] VINCKE P., “Robust solutions and methods in decision-aid”, *Journal of Multi-Criteria Decision Analysis*, vol. 8, no. 8, p. 181–187, 1999.

Chapter 3

The Robustness of Multi-Purpose Machines Workshop Configuration

3.1. Introduction

The aim of this chapter is to propose new tools to assess the configuration robustness of a workshop in uncertain context. The present work is based on a real-life application provided by a semi-conductor manufacturer. It is focused on a photolithography workshop composed of 15 machines which process around 40 different product types. The configuration of this workshop is a major issue because the machines are very expensive, and subject to rapid obsolescence. As a consequence, the workshop manager aims at minimizing idle time on the machines. Thus, the robustness of a configuration is its ability to maintain a loading ratio equal to 100%, even when the actual demand deviates from the forecast demand.

3.2. Problem presentation

In this section, the workshop is modeled as a set of multi-purpose machines and the uncertain context is explicitly delineated. The criterion used to assess the configuration performance and the attached robustness measure are presented afterwards.

Chapter written by Marie-Laure ESPINOUSE, Mireille JACOMINO and André ROSSI.

3.2.1. Modeling the workshop

3.2.1.1. Production resources

A photolithography workshop is composed of m parallel machines that are partially multi-functional: each machine can process a subset of product types. This is due to the fact that processing a product type requires equipping the machine with a resin, which is available in limited quantity. This limitation is the origin of the *qualification constraints*: a machine is said to be qualified for a product type if it is equipped with the required resin, otherwise it is not qualified. Products of the same type require the same resin, and are considered identical. The number of product types is denoted n . The set of all the qualifications of the machines is the workshop configuration. This configuration is the result of a decision-making process aiming at reaching a loading ratio equal to 100%. Furthermore, this choice must be compliant with *technological constraints*: an old-generation machine may not be able to process a new product type (even with the necessary resin), or some different resins may not be installed on the same machine because of incompatibility. For these reasons, the photolithography workshop is a typical example of a set of multi-purpose machines (see [BRU 97]).

In the most general version of the model, the processing speed of machine j for the products of type i is denoted $v(i, j)$, the machines are said to be unrelated. With the notation introduced in Chapter 1, we have $\alpha_1 = R$. Furthermore, $v(i, j)$ is the quantity of products of type i that machine j can process per unit of time. It is necessarily non-negative, and $v(i, j) = 0$ models a technological constraint. It must not be confused with a qualification constraint, as qualification is the result of a choice: when $v(i, j) = 0$, there is no choice to be made as machine j cannot process any product of type i . Qualification arises only if $v(i, j) > 0$.

Preemption and splitting are allowed, so a product type may be processed on several machines simultaneously provided, however, that these machines are qualified for the given product type. Thus, this scheduling problem can be viewed as an allocation problem since the starting times of the products have no impact on the makespan. The set of machines $[1, m]$ is denoted J , and the set of product types $[1, n]$ is denoted I . The qualification matrix Q is an n by m binary matrix such that $Q(i, j) = 1$ if and only if machine j is qualified for product type i , otherwise $Q(i, j)$ is zero. Each column of Q models the qualification of a machine, each row of Q models the machines qualified for a product type. The processing speed matrix v has the same dimensions as Q .

Provided that $v(i, j) = 0$ implies $Q(i, j) = 0$ for all i and j , matrix Q_v is defined by $Q_v(i, j) = Q(i, j) \times v(i, j)$ ($\forall i \in I$) ($\forall j \in J$). This matrix models the machines' processing speed only for the product types for which they are qualified.

A qualification matrix is said to be admissible if and only if it has neither a zero-row nor a zero-column. Indeed, a zero-row shows that products which exist cannot be processed by the workshop. Besides, a zero-column shows a machine that is not qualified for any product type: as a consequence, it should not be considered as a resource for the problem. In the rest of this chapter, only admissible qualification matrices are considered. Moreover, the technological constraints are also assumed to always be such that admissible qualification matrices can be built.

3.2.1.2. Modeling the workshop demand

The products that should be processed the same way in the workshop define a product type. The workshop demand is a column vector denoted N , having n elements. The non-negative real $N(i)$ is the total amount of products of type i to be processed by the workshop. $N(i)$ is not necessarily an integer, as $N(i)$ can be the result of a mean calculation. Demand N is admissible if and only if $N(i)$ is non-negative for all i in I . In the rest of this chapter, only admissible demands are considered.

The following example with $n = 3$ product types and $m = 4$ machines is to be used later to illustrate the results presented in this chapter.

$$Q = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad v = \begin{bmatrix} 1 & 2 & 0.5 & 1.2 \\ 1 & 2 & 0.5 & 0 \\ 0 & 2 & 0.5 & 1.2 \end{bmatrix}$$

so,

$$Q_v = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 1 & 2 & 0.5 & 0 \\ 0 & 2 & 0 & 1.2 \end{bmatrix} \quad N = \begin{bmatrix} 1 \\ 2 \\ 1.7 \end{bmatrix}$$

3.2.2. Modeling disturbances on the data

The context of semi-conductor manufacturing is highly uncertain. New product types are launched very often because of regular advances in new technologies, but launching new production processes often generates a lot of scrap at the beginning.

This is the reason why semi-conductors (microprocessors, memory modules) are often very expensive when they appear on the market, and why they become so cheap when the technologies involved in the processing are well mastered. Scrap is not only an economic loss, it also generates disturbances in production planning. If the photolithography workshop is involved at several stages in the production process of semi-conductors, it is said to be reentrant. Furthermore, the workshop we have been focusing on is part of a manufacturing center dedicated to mass production, as well as to research and development. For that reason, the photolithography workshop produces a great variety of product types which also have very different volumes, so that its actual demand cannot be accurately predicted in advance. On the other hand, the machine qualification process is updated monthly, and must ensure that the chosen configuration fits the demand for the entire month. The configuration cannot be updated dynamically to fit the actual demand as qualification is a very time-consuming process (it requires the machines to be stopped in order to perform tests and setups). Thus, the machine configuration process aims at choosing a configuration that leads to the highest loading ratio for a large set of possible demands.

Disturbances affecting upstream workshops may have different kinds of impacts on the photolithography workshop. If a machine breakdown in an upstream workshop leads to a decrease of the workshop demand, the demand mix may also be changed. In the rest of this chapter, the photolithography workshop forecast demand is denoted N_{ref} , and the actual demand is denoted N . The actual demand N differs from N_{ref} by a quantity denoted dN , due to disturbances in upstream workshops. More precisely, it can be written that $N = N_{ref} + dN$, where dN can have negative entries. Thus, $dN(i) < 0$ means that the quantity of products of type i is less than expected. However, as the actual demand is non-negative, $N_{ref}(i) + dN(i)$ must be positive or zero for each i .

The problem of determining the photolithography workshop configuration under uncertain demand can be stated using the formalism adopted in Chapter 1. The problem \mathcal{P} is to find the best possible configuration for the photolithography workshop. The uncertainties are supposed to impact the demand. Such a demand is a scenario denoted \mathcal{I} in Chapter 1. The configuration Q corresponds to a solution S . The performance of S on the instance \mathcal{I} , denoted $z_{\mathcal{I}}(S)$ is the loading ratio of the machines in the photolithography workshop. Providing performance guarantees is also a key feature here, as the loading ratio is expected to remain equal to 100%. To do so, the robustness criterion denoted R_4 in Chapter 1 is to be minimized, as it assesses the absolute deviation from a fixed level \tilde{z} .

3.2.3. Performance versus robustness: load balance and stability radius

3.2.3.1. Performance criterion for a configuration

A manager's attention must often be focused on the photolithography workshop for several reasons. As new technologies frequently appear in the field, photolithography machines are not only subject to rapid obsolescence, but are also very expensive. That is the reason why this workshop often has a low production capacity, compared to other workshops. As a consequence, the photolithography workshop is often a bottleneck in the production flow, being responsible for WIP (Work In Progress) waves that cause disorder in the whole production process by spreading along the production flow. Thus, the workshop configuration is expected to ensure that all the machines have the highest loading ratio. More precisely, the configuration performance is its ability to guarantee the existence of a balanced load plan which meets the demand; the load plan will be called production plan in this chapter.

3.2.3.2. Robustness

The configuration Q_v is expected to ensure that the machines' loading ratio is equal to 100% for the actual demand $N = N_{ref} + dN$. This should hold on a "neighborhood" of N_{ref} . The performance guarantee offered by Q_v around N_{ref} is measured using the stability radius [SOT 98]. The stability radius assesses the minimum magnitude of a disturbance vector dN being such that a balanced production plan cannot be found for N . Thus, this work can be classified among proactive approaches.

3.3. Performance measurement

The considered performance criterion is the maximization of the machine loading ratio. This criterion is assessed in two stages. In the first stage, the minimization problem of the completion time ([LAW 78]) is solved. A solution to this problem is a production plan. Second, the production plan minimizing the machine workload is computed saved up-on the (minimal) completion time. Both stages are delineated in this chapter.

3.3.1. Stage one: minimizing the maximum completion time

For a given speed matrix v , a configuration Q and a demand N , the problem addressed in the first stage aims at finding a production plan minimizing the completion time. Such a production plan must meet demand while satisfying

the qualification constraints. An optimal solution to this problem is returned in polynomial time by solving a linear program derived from [LAW 78]. A production plan can be written as a matrix R_T of the same size as Q , where $R_T(i, j)$ is the time spent by machine j processing products of type i . Satisfying the qualification constraints implies that $R_T(i, j) = 0$ if $Q(i, j) = 0$ for all i and j . Furthermore, $R_T(i, j)$ is non-negative for all i and j . This problem is modeled as a linear program denoted (LP_1) .

$$(LP_1) : \begin{cases} \text{Minimize } C_{\max} \\ \sum_{j=1}^m R_T(i, j) \times Q_v(i, j) = N(i) \quad (\forall i \in I) \\ \sum_{i=1}^n R_T(i, j) \leq C_{\max} \quad (\forall j \in J) \\ R_T(i, j) \times (1 - Q(i, j)) = 0 \quad (\forall i \in I) \ (\forall j \in J) \\ R_T(i, j) \geq 0 \quad (\forall i \in I) \ (\forall j \in J) \end{cases}$$

The maximum completion time is denoted C_{\max} . The first set of constraints in (LP_1) ensures that the demand is met. The second set of constraints defines C_{\max} as the maximum completion time over all the machines. The last two sets of constraints ensure that the qualification constraints are met, and that the production plan has no negative entries. For real-life instances, an optimal solution is returned in less than one second using an interior point algorithm on an Intel Pentium IV Processor.

DEFINITION 3.1.— *A couple (Q_v, N) is said to be simply balanced if (LP_1) has an optimal solution where all the machines share the same workload, so:*

$$C_{\max} = \sum_{i=1}^n R_T(i, j) \quad (\forall j \in J)$$

EXAMPLE.— Let us consider the following configuration Q_v , and the demands N_1 and N_2 defined by:

$$Q_v = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 1 & 2 & 0.5 & 0 \\ 0 & 2 & 0 & 1.2 \end{bmatrix} \quad N_1 = \begin{bmatrix} 1 \\ 2 \\ 1.7 \end{bmatrix} \quad N_2 = \begin{bmatrix} 1.7 \\ 2 \\ 1 \end{bmatrix}$$

An optimal production plan for (Q_v, N_1) (respectively for (Q_v, N_2)) is denoted by R_{T1} (respectively R_{T2}):

$$R_{T1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.75 & 1 & 0 \\ 0 & 0.25 & 0 & 1 \end{bmatrix} \quad R_{T2} = \begin{bmatrix} 0.5 & 0.6 & 0 & 0 \\ 0.557 & 0.457 & 1.057 & 0 \\ 0 & 0 & 0 & 0.8333 \end{bmatrix}$$

$$C_1 = [1 \ 1 \ 1 \ 1] \quad C_2 = [1.057 \ 1.057 \ 1.057 \ 0.8333]$$

Vectors C_1 and C_2 just display the workload of the machines for both production plans. Thus, it can be observed that (Q_v, N_1) is simply balanced while (Q_v, N_2) is not, machine 4 being underloaded because of a lack of type 3 products.

3.3.2. Computing a production plan minimizing machine workload

After solving (LP_1) , the completion time value is minimal. However, the machine workload need not be minimal as well. The following linear program, denoted (LP_2) , returns a production plan R'_T minimizing the wear of the machines, knowing the numerical value for C_{\max} returned by (LP_1) :

$$(LP_2) : \begin{cases} \text{Maximize} \left(\sum_{j=1}^m Z(j) \right) \\ \sum_{j=1}^m R'_T(i, j) \times Q_v(i, j) = N(i) \quad (\forall i \in I) \\ \sum_{i=1}^n R'_T(i, j) \leq C_{\max} - Z(j) \quad (\forall j \in J) \\ R'_T(i, j) \times (1 - Q(i, j)) = 0 \quad (\forall i \in I) \ (\forall j \in J) \\ R'_T(i, j) \geq 0 \quad (\forall i \in I) \ (\forall j \in J) \\ Z(j) \geq 0 \quad (\forall j \in J) \end{cases}$$

The first set of constraints in (LP_2) is similar to the one in (LP_1) . The second set of constraints expresses the workload for each machine j . This load is $C_{\max} - Z(j)$, where $Z(j)$ is the deviation to C_{\max} . This deviation is non-negative as the completion time must not increase. The maximization criterion is the total time during which the machines are not used (while keeping the completion time to its optimal value).

Thus, the production plan R'_T is such that the machines are used only if necessary. This criterion is relevant for a photolithography workshop, where the managers look forward to rationalizing the machines' engagements. Like (LP_1) , (LP_2) , is solved in less than one second.

DEFINITION 3.2.– A couple (Q_v, N) is said to be strictly balanced if (LP_2) has a zero objective value, so:

$$C_{\max} = \sum_{i=1}^n R'_T(i, j) \quad (\forall j \in J)$$

In the following example, the couple (Q_v, N) is simply balanced because R_T is an optimal solution to (LP_1) for which the machine loading ratio is equal to 100%. However, it is not strictly balanced as R'_T , an optimal solution for (LP_2) is such that all the machines do not have the same workload.

$$\begin{aligned} v &= \begin{bmatrix} 2 & 0 & 4 \\ 1 & 2 & 2 \\ 4 & 1 & 1 \end{bmatrix} & N &= \begin{bmatrix} 8 \\ 3 \\ 5 \end{bmatrix} & Q_v &= \begin{bmatrix} 0 & 0 & 4 \\ 1 & 2 & 2 \\ 4 & 1 & 1 \end{bmatrix} \\ R_T &= \begin{bmatrix} 0 & 0 & 2 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} & R'_T &= \begin{bmatrix} 0 & 0 & 2 \\ 0 & 1.5 & 0 \\ 1.25 & 0 & 0 \end{bmatrix} \\ C &= \begin{bmatrix} 2 & 2 & 2 \end{bmatrix} & C' &= \begin{bmatrix} 1.25 & 1.5 & 2 \end{bmatrix} \end{aligned}$$

By construction, any strictly balanced couple is also simply balanced, but the reverse is not true.

Then, the production plan R_T will be said to be simply (respectively strictly) balanced if it is an optimal solution to (LP_1) (respectively to (LP_2) , i.e. $Z = 0$).

3.3.3. The particular case of uniform machines

The present section aims to show that when the photolithography workshop can be modeled with uniform machines, any simply balanced production plan is also strictly balanced. Thus, simple balance is sufficient since strict balance goes hand in hand with simple balance in that particular case.

Let v_p be a column vector having n strictly positive elements, and v_m be a row vector having m strictly positive elements. It is recalled that if matrix v is defined by $v = v_p \times v_m$, it is a rank-one matrix and so it models a uniform machines workshop.

THEOREM 3.1.– *If the workshop is modeled with uniform machines (i.e. matrix v is defined by $v = v_p \times v_m$), then any simply load balanced couple (Q_v, N) is also strictly balanced.*

The proof of this theorem can be found in [ROS 03].

3.4. Robustness evaluation

The performance of a configuration is its ability to guarantee that a strictly balanced production plan exists for the actual demand N . Such a performance indicator is a binary function, as no deviation from strict balance is allowed by the user. In that case, the robustness of a configuration can be assessed through the magnitude of the disturbances on the demand for which the production plan remains strictly balanced.

In the next section, the theoretical background used to determine all the demands for which there exists a balanced production plan is presented. An appropriate robustness measure for a configuration in the neighborhood of a given demand is derived from that theory.

3.4.1. *Finding the demands for which the production plan is balanced*

There exist non-balanced production plans only because of non-qualifications (i.e. zero entries in Q). Thus, some overloaded machines cannot share their load with the other machines because of non-qualifications. These non-qualifications are the result of economic choices or technological constraints. The main purpose of the next paragraph is to characterize all the demands for which there exists a balanced production plan for a given configuration Q_v . This set of demands is denoted $B(Q_v)$.

Determining $B(Q_v)$ is a two-stage process. The first stage aims at finding all the maximum rectangles of zeros in Q_v . In the second stage, a linear constraint is built from each of the maximum rectangles of zeros returned in the first stage. Then, each constraint is shown to be associated with a frontier of $B(Q_v)$, which is the result of the second stage.

Stage 1: finding all the maximum rectangles of zeros in Q

Let Q be a configuration and v a speed matrix having n rows and m columns. I_k denotes a subset of I , and J_k a subset of J .

DEFINITION 3.3.– *The Cartesian product $I_k \times J_k$ is a rectangle of zeros in Q if and only if:*

$$Q(i, j) = 0 \quad (\forall (i, j) \in I_k \times J_k)$$

DEFINITION 3.4.– *The Cartesian product $I_k \times J_k$ is a maximum rectangle of zeros in Q if and only if there does not exist any other rectangle of zeros $I'_k \times J'_k$ including $I_k \times J_k$.*

The algorithm used to find all the maximum rectangles of zeros in Q is derived from the work of [NOU 99], and addresses issues related to Galois lattices. The maximum rectangles of zeros in Q can be viewed as concepts in the sense of non-qualification. The set of these concepts is the Galois lattice edges associated with the context (i.e. the configuration Q). Nourine and Raynaud propose an algorithm to compute these edges, by building the lexicographic tree of a family denoted F generated by a basis denoted B . This basis has m elements, element j being the set of product types for which machine j is qualified.

Family F is a finite list of sets of product types that is stable for union. Each element f of the family F is associated with the set $\gamma(f)$, made of all the machines that are qualified for product types in f only. Then, F is the list of the set of product types $\{I_1, I_2, \dots, I_h\}$, and $\gamma(F)$ is the associated list of the corresponding machines $\{J_1, J_2, \dots, J_h\}$. The non-negative integer h is defined by $h = |F| = |\gamma(F)|$, it is also the number of maximum rectangles of zeros that can be found in Q .

The neutral element for union is denoted e . Here follows the algorithm proposed by [NOU 99] to build the lexicographic tree on F , that is to say all the couples $(f, \gamma(f)) = (I_k, J_k)$.

Initialization: $F = e$ and $\gamma(F) = \{e\}$

For each element b in the basis B , do

 For each element f in F , do

$f' = f \cup b$

 If $f' \notin F$ then do

$F = F \cup \{f'\}$

$\gamma(f') = \gamma(f) \cup \{b\}$

 End-If

 End-For

End-For

Nourine and Raynaud have shown that building the lexicographic tree from B requires $o((n + m) \times m \times h)$ operations. Furthermore, they state that determining h without building the lexicographic tree is still an open problem.

Stage 2: generating $B(Q_v)$ for uniform machines

All the maximum rectangles of zeros of Q are assumed to be known. The h sets $I_k \times J_k$ (with $1 \leq k \leq h$) can be deduced by setting $I_k = f_k$ and $J_k = \gamma(f_k)$.

DEFINITION 3.5.– *$I_k \times J_k$ is a maximum rectangle of zeros of Q , and we denote $I'_k = I \setminus I_k$ and $J'_k = J \setminus J_k$. The constraint (C_k) is defined by:*

$$(C_k) : \sum_{i \in I_k} \frac{N(i)}{v_p(i)} \leq \frac{\sum_{j \in J'_k} v_m(j)}{\sum_{j \in J_k} v_m(j)} \times \sum_{i \in I'_k} \frac{N(i)}{v_p(i)}$$

With this constraint, we express the fact that a maximum rectangle of zeros implies that the load of the machines that do not belong to J_k for the products of type I_k must be less than the load of the machines belonging to J_k for the products of type I'_k . The border (F_k) which is associated with this constraint can be written as follows:

$$(F_k) : \sum_{i \in I_k} \frac{N(i)}{v_p(i)} = \frac{\sum_{j \in J'_k} v_m(j)}{\sum_{j \in J_k} v_m(j)} \times \sum_{i \in I'_k} \frac{N(i)}{v_p(i)}$$

Let us note that the expressions (C_k) and (F_k) are valid only for uniform machines. Such a constraint limits the load of some machines if the load-balance property is to be enforced. More details can be found in [ROS 03].

THEOREM 3.2.– *The demand N is balanced if and only if N satisfies the constraints (C_k) , for all k in $[1, h]$.*

The proof of this theorem is given in [ROS 03]. However, for a better understanding, this theorem is illustrated in an example. The matrix Q_v previously defined has 4 rectangles of zeros. $v_p = [1 \ 1 \ 1]^T$ and $v_m = [1 \ 2 \ 0.5 \ 1.2]$. $B(Q_v)$ is characterized by the $h = 4$ following constraints:

$$Q_v = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 1 & 2 & 0.5 & 0 \\ 0 & 2 & 0 & 1.2 \end{bmatrix}$$

$$B(Q_v) : \begin{cases} N(1) \leq \frac{3}{1.7} \times (N(2) + N(3)) & I_1 = \{1\} \quad J_1 = \{3, 4\} \\ N(1) + N(2) \leq \frac{3.5}{1.2} \times N(3) & I_2 = \{1, 2\} \quad J_2 = \{4\} \\ N(3) \leq \frac{3.2}{1.5} \times (N(1) + N(2)) & I_3 = \{3\} \quad J_3 = \{1, 3\} \\ N(1) + N(3) \leq \frac{4.2}{0.5} \times N(2) & I_4 = \{1, 3\} \quad J_4 = \{3\} \end{cases}$$

The first constraint can be explained as follows. The rectangle of zeros $I_1 \times J_1$ represents the fact that type 1 products are not qualified on machines M_3 and M_4 . The time spent by machines M_1 and M_2 to process all the type 1 products must be less than the time spent by machines M_3 and M_4 processing all the other types of products (i.e. types 2 and 3). If this was not the case, the load of machines M_1 and M_2 would be greater than the load of machines M_3 and M_4 . That is to say:

$$\frac{N(1)}{1+2} \leq \frac{N(2)+N(3)}{0.5+1.2}$$

and this is the first constraint of $B(Q_v)$.

3.4.2. Stability radius

In the previous sections, all the balanced demands for a given qualification matrix associated with a matrix speed have been characterized. At the beginning of every period, the forecast demand of the photolithography workshop is supposed to be known. This forecast demand denoted N_{ref} is computed from the customer's demand for electronic components. However, the actual demand may be rather different. This is due to machine breakdowns and more generally to disturbances that affect other workshops. While it seems easy to check if the forecast demand leads to a balanced production plan or not, this may not be obvious for an unknown demand. In this section, we propose a measure of robustness to evaluate the minimum distance between the forecast demand and the actual demand that may lead to the loss of the load-balancing property. The notion of stability radius defined in [SOT 98] is used for this purpose.

DEFINITION 3.6.– Let:

- p be a demand in \mathbb{R}^q (for modeling a problem instance),
- z be an optimal solution for some instance (the set of decisions made for an instance),

Then the closed ball $O_r(p)$ of radius r and center p in the space of dimension q is a ball of stability of z if, for any vector p' in the intersection of $O_r(p)$ and \mathbb{R}^q , the solution z remains optimal. The maximum value for the radius r of a ball of stability $O_r(p)$ associated with the solution z is defined as the stability radius of z .

Replacing \mathbb{R}^q with \mathbb{R}^n , p with N_{ref} , z with Q_v and choosing the load-balance property as a criterion is performing an adaptation of the definition to the photolithography workshop.

We use this tool to characterize a set of demands for which a configuration Q guarantees the load balancing property. This set is defined by the neighborhood of the forecast demand N_{ref} for which the solution is balanced.

DEFINITION 3.7.— *Let us define $\Omega = (\omega_k)$ the vector of distances between N_{ref} and the h borders of $B(Q_v)$. The process for determining those distances are not presented here; more details can be found in [ROS 03].*

DEFINITION 3.8.— *We define $r(Q_v, N_{ref})$ as the following quantity:*

$$r(Q_v, N_{ref}) = \min_{k \in [0, h]} (\omega_k)$$

THEOREM 3.3.— *If $r(Q_v, N_{ref}) \geq 0$, then it is the stability radius of the configuration matrix Q associated with the speed matrix v and a forecast demand N_{ref} , and any demand N defined by $N = N_{ref} + dN$ with $|dN|_1 \leq r(Q_v, N_{ref})$ leads to a balanced production plan.*

This theorem has been shown in [ROS 03]. The L1-norm has been chosen because it measures the total load difference between two demands.

The set of the demands N defined above is a neighborhood of N_{ref} for which the load balancing property is guaranteed. However, there can exist some balanced demands N for which $|dN|_1 > r(Q_v, N_{ref})$ (but these demands are not characterized). This shows that the stability radius is a synthetic but restrictive measure of robustness: it gives a sufficient but not necessary condition of robustness.

3.4.3. Graphic representation

Let us consider Q_v , $B(Q_v)$ and N_1 defined in section 3.3.1. For the sake of clarity, it is assumed that the variations dN which affect N_{ref} are such that

$$\sum_{i=1}^n dN(i) = 0.$$

The demand space is of dimension $n = 3$. Let us define H_s as the set of the demands N defined by $N = N_1 + dN$ shown in Figure 3.1.

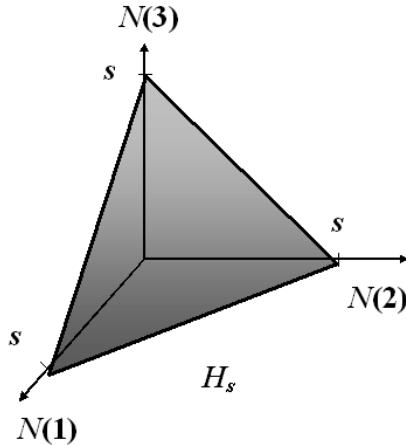


Figure 3.1. H_s

Such an assumption on dN leads to the simplification of the constraints of the previous academic example for the definition of $B(Q_v)$. The new set of constraints is denoted $B_1(Q_v)$, and defined below in H_s :

$$B_1(Q_v) : \begin{cases} N(1) \leq 3 \\ N(1) + N(2) \leq 3.5 \\ N(3) \leq 3.2 \\ N(1) + N(3) \leq 4.2 \end{cases} \quad \Omega(Q_v, N_1) = \begin{bmatrix} 4 \\ 1 \\ 3 \\ 3 \end{bmatrix}$$

where ω_k is the k th component of the vector Ω . This new set of constraints is returned using the following equality, which expresses the conservation of the load of N :

$$\sum_{i \in I} N(i) = \sum_{i \in I} N_1(i) = s$$

where s is a strictly positive number.

We can also replace N_1 with N'_1 defined by $N'_1 = \frac{N_1}{\sum_{i \in I} N_1(i)}$. Indeed, the balance property still holds when multiplying the demand by a strictly positive number.

$B(Q_v)$ is represented by the tinted zone in Figure 3.2. It can be observed that N_1 belongs to $B(Q_v)$, while this is not the case for N_2 . Ω is the vector of distances from N_1 to each of the border of $B(Q_v)$. The arrow represents the stability radius $r(Q_v, N_1)$ in H_s : that is the smallest distance from N_1 to the four borders of $B(Q_v)$.

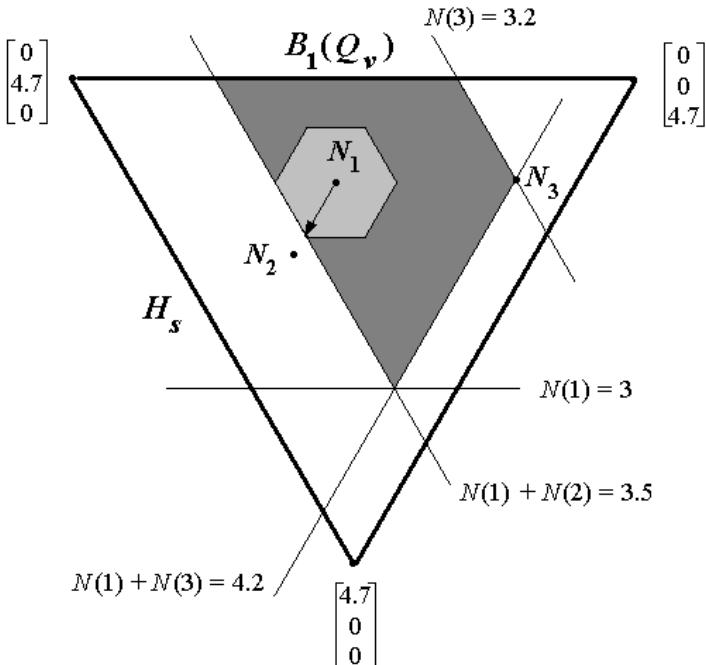


Figure 3.2. Stability radius

In this example, $r(Q_v, N_1) = 1$. The set of demands defined by $B_1(Q_v)$ is represented in dark gray. The light gray regular hexagon is included in $B_1(Q_v)$: it is the neighborhood of N_1 defined by the stability radius, that is to say the set of demands N defined by $N = N_1 + dN$ with $|dN|_1 \leq r(Q_v, N_1)$.

The stability radius is represented by the arrow. The hexagon is the ball of radius equal to one for which N_1 is the center. This ball is not circular since the L1-norm is used instead of the L2-norm. More precisely, the hexagon is the intersection between the ball and H_s .

3.5. Extension: reconfiguration problem

In this section, the stability radius is used as a measurement tool for helping the decision-maker with improving the configuration quality. The so-called quality used here is robustness, defined as the configuration ability to cope with demand uncertainties while maintaining the load balance property for production plans. To do so, it is shown how to add qualifications to the configuration in order to increase its robustness.

3.5.1. Consequence of adding a qualification to the matrix Q

The set $B(Q_v)$ of balanced demands is defined by the constraints (C_k) which are based on the maximum rectangles of zeros in the matrix Q . The aim of this section is to study how $B(Q_v)$ and the stability radius are modified when a qualification is added (a one entry in the matrix Q).

It is assumed that $Q(i_0, j_0) = 0$, this zero belonging to the maximum rectangle of zeros $I_k \times J_k$. Q_+ is the configuration matrix defined by $Q_+(i, j) = Q(i, j)$ for all $i \neq i_0$ and for all $j \neq j_0$, and $Q_+(i_0, j_0) = 1$. Knowing the rectangles of zeros in Q , the new configuration Q_+ is now considered. Consequently, constraint (C_k) is replaced with two new constraints denoted (C'_k) and (C''_k) :

$$(C'_k) : \sum_{i \in I_k - \{i_0\}} N(i) \leq s - \sum_{j \in J_k} v_m(j)$$

$$(C''_k) : \sum_{i \in I_k} N(i) \leq s - \sum_{j \in J_k - \{j_0\}} v_m(j)$$

If $|I_k| = 1$ or $|J_k| = 1$ then only one constraint has to be considered since the second one turns out to be always true.

Now, we consider the distances from N_{ref} to the new borders of $B(Q_v)$ induced by (C'_k) and (C''_k) .

$$\omega'_k = 2 \times \left(s - \sum_{j \in J_k - \{j_0\}} v_m(j) - \sum_{i \in I_k} N_{ref}(i) \right) = \omega_k + 2 \times v_m(j_0)$$

$$\omega''_k = 2 \times \left(s - \sum_{j \in J_k} v_m(j) - \sum_{i \in I_k - \{i_0\}} N_{ref}(i) \right) = \omega_k + 2 \times N_{ref}(i_0)$$

The last equalities show that N_{ref} is much farther from each of the new constraints than it was from (C_k) , so the stability radius does not decrease. Let us define $G(Q_v, N_{ref}, k, i_0, j_0)$ by:

$$G(Q_v, N_{ref}, k, i_0, j_0) = \min (\omega'_k - \omega_k, \omega''_k - \omega_k)$$

Furthermore, setting $Q(i_0, j_0)$ to 1 could change other constraints with the same consequences. More precisely, all the constraints coupled with a maximum rectangle of zeros which contains $Q(i_0, j_0)$ are modified like (C_k) .

The strategy for adding a new qualification in the current configuration matrix is thus the following one. The h constraints of $B(Q_v)$ are considered by increasing order of the distances to N_{ref} . In order to maximize the stability radius, the new qualification has to belong to the intersection of the closest constraints to N_{ref} .

3.5.2. Theoretical example

$$Q_v = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 1 & 2 & 0.5 & 0 \\ 0 & 2 & 0 & 1.2 \end{bmatrix} \quad N_1 = \begin{bmatrix} 1 \\ 2 \\ 1.7 \end{bmatrix} \quad \Omega(Q_v, N_1) = \begin{bmatrix} 4 \\ 1 \\ 3 \\ 3 \end{bmatrix}$$

In this example, only one critical constraint has to be considered since the vector $\Omega(Q_v, N_1)$ has only one minimum equal to 1. This minimum is reached for the constraint (C_2) : $N(1) + N(2) \leq 3.5$. Thus, adding only one qualification is sufficient to increase the stability radius. This one entry must be added into the maximum rectangle of zeros of Q that defines the critical constraint $I_2 \times J_2$ with $I_2 = \{1, 2\}$ and $J_2 = \{4\}$. Thus, there are two possibilities as $|I_2 \times J_2| = 2$. The closest constraints from N_1 after (C_2) are (C_3) and (C_4) . The intersection between the corresponding rectangles of zeros and $I_2 \times J_2$ is empty, thus whatever the zero entry in $I_2 \times J_2$ that is replaced with one, the increase of the stability radius will be the same.

For example, setting $Q(1, 4)$ to one yields:

$$Q_+ = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

$$B(Q_{v+}) : \begin{cases} N(1) + N(3) \leq 4.2 \\ N(2) \leq 3.5 \\ N(3) \leq 3.2 \end{cases} \quad \Omega(Q_{v+}, N_1) = \begin{bmatrix} 3 \\ 3 \\ 3 \end{bmatrix}$$

Thus it can be seen that $r(Q_{v+}, N_1) = 3$. The stability radius would be the same if we had set $Q(2, 4)$ to one. However, adding a qualification outside $I_2 \times J_2$ does not allow us to increase the stability radius.

3.5.3. Industrial example

Let us consider an industrial photolithography workshop with 36 types of products and 13 machines. The forecast demand N_{ref} is about 9,000 units of products dispatched into the 36 product types. The stability radius was initially equal to 43.2 for the original configuration (which was given). This means that the original configuration was able to maintain the load balance property provided that the demand variation magnitude $|dN|_1$ is less than 43.2 products. This is quite a low value in an uncertain context. After adding only one qualification, the stability radius increases by 1,360%, reaching 587.4. Thus, the robustness of the configuration increases in the same proportions as the load balance property is guaranteed for any demand variation magnitude up to 587.4 products.

3.6. Conclusion and perspectives

In this chapter, two measures of robustness were presented for the configuration of multipurpose machines in an industrial context. The method proposed to increase the robustness of a configuration is based on the evaluation of the robustness of this configuration Q . To do this, we determine $B(Q_v)$ which models the flexibility structure of the configuration Q for the forecast products mix. It is a complete piece of information about Q which allows us to determine all the demands for which the configuration guarantees the desired performance (i.e. a perfect load balance). The second proposed measure for robustness uses the information returned by $B(Q_v)$. For a forecast demand, the stability radius measures the maximum demand variation magnitude for which the desired performance is guaranteed.

The presented approach does not take into account some industrial constraints (like the limited availability of resin for instance). Nevertheless, a set of action (to increase robustness) is proposed to the user, who can take into account such constraints. Furthermore, it is possible to use this method for uniform machines for which photolithography is an application among others.

Addressing the attached scheduling problem is an interesting perspective as far as the way of consuming products in the workshop could have an impact on the

robustness (due to the variations of N). Similarly, we can generalize this study by considering setup costs, which do not exist in the photolithography workshop, but may be relevant in some other application fields.

3.7. Bibliography

- [BRU 97] BRUCKER P., JURISH B. and KRÄMER A., “Complexity of scheduling problems with multi-purpose machines”, *Annals of Operations Research*, vol. 70, p. 57–73, 1997.
- [LAW 78] LAWLER E. and LABETOULLE J., “On preemptive scheduling of unrelated parallel processors by linear programming”, *Journal of the Association for Computing Machinery*, vol. 25, p. 612–619, 1978.
- [NOU 99] NOURINE L. and RAYNAUD O., “A fast algorithm for building lattices”, *Information Processing Letters*, vol. 71, p. 199–204, 1999.
- [ROS 03] ROSSI A., *Ordonnancement en milieu incertain, mise en œuvre d'une démarche robuste*, PhD Thesis, Institut National Polytechnique de Grenoble, 2003.
- [SOT 98] SOTSKOV Y., WAGELMANS A. and WERNER F., “On the calculation of the stability radius of an optimal or an approximate schedule”, *Annals of Operations Research*, vol. 83, p. 213–252, 1998.

Chapter 4

Sensitivity Analysis for One and m Machines

The problems studied in this chapter are scheduling problems on a single machine, and on a fixed number of parallel machines. Uncertainty is either on task execution times or on communication durations between tasks, if any. In one particular case, machine unavailability are also investigated.

The study considers a proactive framework, in the sense of Chapter 1: the scheduling algorithms are used before execution, but they take uncertainty into account. Anyway, it is assumed that temporal flexibility is permitted during the execution phase. Hence, the result of the algorithms is not a fixed schedule, but a task sequence on the machine, or on each machine. The set of these sequences is called a solution throughout the chapter or, by a slight abuse of notation, a schedule. The proactive approach assumes at least a partial knowledge of the data uncertainties, together with the use of that knowledge in the proposed solution choice. Consequently, the desired solutions are those with good performances with regard to some of the robustness measures proposed in this book's introduction (*robust* solutions). The aim of the chapter is to show the interest and the limits of sensitivity studies for building such solutions. Not all existing works are presented here. We rely on some significant publications in the field of sensitivity analysis for single and m machine problems.

Chapter written by Amine MAHJOUB, Aziz MOUKRIM, Christophe RAPINE and Eric SANLAVILLE.

The chapter is organized as follows. Sensitivity analysis (SA) is clearly defined in section 4.1; its traditional use for varied combinatorial optimization problems is presented, with the questions arising from this use. Then several SA are presented for single machine problems. In one case, the SA clearly shows the lack of robustness of a statically (all parameters fixed) optimal solution. A systematic search of robust solutions is then proposed. The m machine without communication delay problems are studied with uncertain execution times. A more detailed analysis is presented in one case where all durations are uncertain, and for which the relative deviation in the worst case (also described as the competitiveness ratio) can be computed as a function of the disturbance's magnitude. The case of uncertain communication delays is presented last. A detailed study is done in the two machine case, and general results are proposed.

4.1. Sensitivity analysis

It might not be useless to think about the use of SA for decision problems in uncertain context. In Chapter 1, the *tool* aspect of SA is highlighted. Through an *a priori* study, it makes it possible to decide about an algorithm or a solution quality in an uncertain context. Anyway, papers considering SA are numerous, their often implicit definitions of SA might be very different. One definition of sensitivity analysis might be the following.

DEFINITION 4.1.– Consider an optimization problem \mathcal{P} and \mathcal{I} an instance of \mathcal{P} . Let S^* and z^* be an optimal solution for \mathcal{I} and its value, respectively. A sensitivity analysis on \mathcal{P} and \mathcal{I} consists of answering (at least partially) to the following questions:

- 1) In what neighborhood of \mathcal{I} does S^* (resp. z^*) remain optimal?
- 2) In what neighborhood of \mathcal{I} does S^* remain feasible, with acceptable performance?
- 3) Considering \mathcal{I}' a neighbor of \mathcal{I} , is S^* feasible for \mathcal{I}' and then, what is its performance degradation?
- 4) Considering \mathcal{I}' a neighbor of \mathcal{I} , what is the new optimal solution (resp. value)?

and to the question variants obtained by replacing S^* by an arbitrary solution in questions 2 and 3.

Hopefully, this definition is general enough, so that any other (reasonable) definition from an SA study is included in it, in particular because of the generality of the notions of solution and neighborhood. We always start from a given instance, which is perturbed. In the first case, the goal is to characterize a set of “acceptable” disturbances (questions 1 and 2), in the second case (questions 3 and 4) the

disturbance is imposed. Question 4 is exactly equivalent to the reoptimization problem.

The main literature stream comes from the linear programming sensitivity analysis. The aim is to answer questions 1 and 2 when one parameter is allowed to vary (right hand side or objective coefficient). The answer is easy to obtain for one parameter, hence many studies arose for more complex model, as integer linear programming. These works are justified by the fact that SA would be of great help for optimization under uncertainty, but this can be questioned.

Wallace [WAL 00] proposes a quite corrosive analysis of SA use, citing in particular two representative books, Gal and Greenberg [GAL 97] and Ravindran *et al.* [RAV 87]. His study considers a large framework for decision aid, but by hypothesis a probabilistic model can be used, for instance by associating a probability with each scenario. The probabilistic model can nevertheless remain implicit. For Wallace, SA should help to build one or several “robust” solutions, that is, close to the optimum for $E(z)$: according to Chapter 1, in the sense of the expected stability measure with respect to the optimization criterion z .

As we defined it, SA (except for question 4) is for Wallace a *parametric analysis*. Thanks to such analysis, it should be possible to choose one or several stable solutions, that is, solutions which remain optimal for a reasonably large set of disturbances. To build such solutions, we may also consider a sample, for instance randomly generated, of possible scenarios. Best solutions would be obtained from the different optimal solutions, by keeping their common structure. Unfortunately, Wallace shows that in many cases, the set of proposed solutions (obtained either by a parametric study or by sampling) does not contain the optimal solution for $E(z)$. This robust-optimal solution might even miss the structure common to all proposed solutions. Indeed, these common features appear because of the deterministic nature of the models they come from. Wallace entails that deterministic analysis (where random distributions are not explicitly associated with the uncertain parameters) are useless for the building of optimal solutions according to a probabilistic criterion (what is true for the expectation remains true for stochastic optimization, among others). Some methods or decisions that are more flexible in the sense of this book will often be discarded, as they are suboptimal for each scenario. The method implicitly promoted by Wallace consists of using a probabilistic framework, even if the model remains an approximation. This model may be used either to build optimal solutions according to traditional stochastic criteria as stochastic minimization, or simply to legitimate the choice of a fixed domain for the parameter values,

hence permitting a worst case study. SA based on a parametric analysis would be useful only for answering precise questions such as: “What if this parameter value is changed by exactly this amount?”, for a model where the studied parameter is originally completely deterministic. Wallace’s point of view is now widely shared in the community of optimization under uncertainty. Recently, a number of works consider explicitly robust optimization, particularly in linear programming (see seminal works by Soyster [SOY 73], Ben-Tal and Nemirovski [BEN 99], Bertsimas and Sim [BER 03] and Gabrel *et al.* [GAB 07] for some recent results).

Let us now examine the Wallace analysis in the specific field of scheduling under uncertainty. He considers two phase decision models (extensible to n phases), uncertainties being progressively removed. This is very similar to the model used throughout this chapter: one off-line phase, one on-line phase, particularly when a proactive or proactive/reactive method is mandatory (for instance, for problems dealt with in this chapter, it is often necessary to choose off-line the machine allocation). Hence its conclusions are valuable for us. Let us note, however, that he considers a stochastic model to encompass uncertainties; furthermore, the study looks for optimal solutions with respect to some stochastic criterion, especially expectation. When robust solutions are sought, there are many ways to measure robustness as shown in Chapter 1. A solution that optimizes the criterion optimization might be rejected if its performance variations are too large. By contrast, worst case analysis, together with appropriated robustness measures, make it possible to build “naturally” robust solutions. The difficulty here consists of defining what the worst case is! Some gross probabilistic model may help us here, by limiting the instances or scenarios with rather small probability, or by making some hypotheses on the possible instances (for instance, no more than one machine breakdown at a time).

What is well shown in [WAL 00] is that a stability criterion (like measuring the size of the region where the proposed solution remains optimal) is not enough in general to build robust solutions. However, traditional, parametric SA, issued from linear programming, does not leave this framework. Hence it is only useful in the intrinsically deterministic case, where few disturbances might occur. The works of Sotskov and several co-authors (see for instance Sotskov, Wagelmans and Werner [SOT 98]) are of that kind. Their main theoretical contribution consists of defining the stability radius (an answer to question 1), and in proposing methods to compute that radius. However, this definition is not specific to scheduling.

DEFINITION 4.2.– [SOT 98] *Considering a problem \mathcal{P} , an instance \mathcal{I} and an optimal solution s for \mathcal{I} , the stability radius $\rho_s(\mathcal{I})$ is the maximum radius of a ball centered on \mathcal{I} , inside which s remains optimal.*

In Sotskov *et al.* the radius is defined on execution duration only, and \mathcal{I} is identified with a vector p of execution durations. The infinite norm is used, but the stability radius can easily be adapted to other contexts. The general results presented in [SOT 98] are necessary and sufficient conditions, either on the existence of a strictly positive radius or for the case of an infinite radius (a solution remains optimal for any instance). They also provide general formulae for the computation of ρ , but this computation is of exponential complexity. Their work applies to a large set of shop scheduling, but not to parallel machines, except if machine allocation is fixed. Even if these results are difficult to exploit directly, the computation of ρ for a given problem and a specific algorithm can provide useful information. Examples of such computations may be found in Hall and Posner (see section 4.3.1) and Chapter 3. Note again that stability radius is useful in the context of small disturbance. Sotskov *et al.* also introduced the approximated stability radius that gives the maximum magnitude of a disturbance for a fixed loss of performance (a variant of question 3). Computing this approximated radius is only possible in some particular cases.

In the case of a true uncertainty on a set of parameters, there is no interest in starting from estimated values (that is, mean values), and performing an analysis for the variation of one parameter only. The objective is at least to study the impact of simultaneous variation around their mean value. Because of the complexity of stochastic models, it is extremely difficult to look for an optimal solution relative to some stochastic criterion, especially if unstable solutions (having a bad worst case behavior) should be avoided. Such a search is to be avoided anyway, if our knowledge on the data is too incomplete. Last, we should observe that unlike questions 1 and 2 from definition 4.1, the answer to question 3 will provide more reliable data on the robustness of a solution or of a method. A natural way of investigation consists of starting from a method or a solution (for instance, optimal in a deterministic setting), and to check its robustness.

It is therefore logical to make sensitivity analysis of the worst case type on particular algorithms, or on particular solutions, thus generalizing question 3: a whole neighborhood of initial instance \mathcal{I} is considered. Wallace's observations do not question the interest of such analysis, but the limits on the parameters variations arise as a major issue. Presenting the results as a function of the disturbance maximum magnitude is a possible answer, see Penz *et al.* [PEN 01], section 4.3.2. Hence the decision maker will get a more global vision of the expected performances. The possibility of working on a probabilistic model should be considered, the choice depending on our knowledge of the problem data. This probabilistic hypothesis is the starting point of several chapters of this book including Chapters 5, 6 and 7.

4.2. Single machine problems

4.2.1. Some analysis from the literature

Classical scheduling problems with uncertainties have recently been studied. Chanas and Kasperski [CHA 04] provide a SA in the classical sense (answer to question 1) for problem $1\| \max w_j L_j$. The problem is solved by the polynomial Lawler algorithm, but many schedules are optimal. For a given schedule and for each parameter (duration, due date or weight) a set of values is computed for which the schedule is optimal. This set is a union of intervals, which is computed in linear time. Contrary to linear programs, their method generalizes easily to the case when several parameters may vary. Then the set of possible values is the union of polyhedra. [CHA 04] does not provide any result on the choice of the less sensitive deterministic-optimal schedule. However, Kasperski in [KAS 05] gives a polynomial algorithm to compute the most robust schedule in the minmax regret sense for $1\|L_{\max}$, when all parameters vary within intervals.

Hall and Posner [HAL 04] possibly provide the most significant study: they examine the important questions for sensitivity analysis in scheduling, the proposed examples considering single or m machine problems. Single machine specific results are mentioned here. In some examples, such as $1\| \sum w_j C_j$, algorithms are provided to compute the maximal variations for each parameter for which S^* remains unchanged, but also, if needed, the new optimal solution (questions 1 and 4 simultaneously). These algorithms are more efficient than simple re-optimization. Their complexity depends on the number of modified parameters, but also on the position in the schedule of the tasks concerned by the change.

In the case of pseudo-polynomial problem $1\| \sum w_j U_j$, whose deterministic version can be solved by dynamic programming, two types of results are proposed. First, Hall and Posner show that one might be done simultaneously with the computation of an optimal solution, at the price of additional computation (the algorithm remains pseudo-polynomial). There are two new dynamic programming algorithms, according to the fact that the disturbances are on the durations or on the weights. They allow us to know *a priori* the disturbances for which the computed solution remains optimal. Hall and Posner [HAL 04] observe that a large number of optimal solutions exist. It is also possible to choose among those solutions the most robust with regard to disturbances on weights (and similarly on durations). The robustness measure used here is simply the size of the smallest disturbance (absolute or relative) for which the solution is no longer optimal. In other words, the

most robust solution minimizes the stability radius (stability measure relative to the performance criterion). The next section studies the same problem (without weights) with uncertainty on the machine availability.

Daniels and Kouvelis [DAN 95] on the one hand and Lebedev and Averbakh [LEB 06] on the other hand study problem $1\| \sum C_j$, but their studies begin where Hall and Posner left off; they are looking for the most robust solution with respect to the worst case absolute deviation (also called largest regret), defined in Chapter 1 by $\max_{\mathcal{T}} z_{\mathcal{T}}(S) - z_{\mathcal{T}}^*$. [DAN 95] shows that the problem is NP-complete as soon as two scenarios only are considered for the execution durations. [LEB 06] assumes that each duration is in a given continuous interval. The problem is still NP-complete unless intervals have the same center and the number of tasks is even! Remember that the list SPT algorithm is optimal for the corresponding deterministic problem. There is no SA for SPT in the case of arbitrary intervals with this measure (it is immediate, anyway, that its stability ratio is null if two tasks have the same estimated duration). In section 4.3.2 such SA is performed for relative deviation (analysis valid for one or m machines). In the centered interval case all tasks have the same average duration, hence any order is SPT. The optimal algorithm of [LEB 06] consists of placing in the middle of the sequence the tasks with the largest interval.

4.2.2. Machine initial unavailability for $1\| \sum U_j$

The results of this part are extracted from [MAH 04]. The problem of minimizing the number of late jobs is considered. Disturbances arise from a potential machine unavailability that may occur at the beginning of the sequence. The problem has two interesting properties. On the one hand, the deterministic variant is polynomial and may be solved in $\mathcal{O}(n \log n)$. As we shall see, the optimal solution is extremely sensitive to disturbances: there is no sensitivity guarantee that depends only on the unavailability period duration. Hence it is important to look for robust solutions; the worst case absolute deviation is used to measure that robustness. On the other hand, and contrary to previous problem $1\| \sum C_i$, which is NP-complete with only two scenarios, under some hypotheses a robust solution may be found in polynomial time, even when the disturbance is inside a real interval $[0, S]$, and not only in a finite set Ω of scenarios.

4.2.2.1. Problem presentation

A set of n tasks must be executed sequentially, without preemption, on one single resource (machine). Each task j has execution time p_j and deadline d_j . The goal is to minimize the number of late tasks, that is, finishing after their deadline.

To illustrate this problem, consider a firm that receives orders, each order being one or a set of tasks to deliver before a time limit imposed by the customer. In the production process, each task has to pass through a “bottleneck” machine. To decide if an order should be accepted, the firm schedules the received orders and rejects those that cannot be satisfied in time. This problem is polynomially solved by the Hoodson and Moore (*HM*) algorithm, which is a slight modification of *EDD* (*Earliest Due Date*). Tasks are ordered by increasing due date. If a task j is late, the largest task scheduled before j finishing time is *disqualified*. The disqualified task will be executed after all qualified tasks (this is equivalent to reject the customer’s order). The algorithm is detailed in [BŁA 93].

Let us now assume that the machine might be unavailable at the beginning of the production process (during time interval $[0, s]$). This unavailability might occur if the bottleneck machine broke down, or if a component was delivered late. The effective duration unavailability s is only known *a posteriori*, that is, after the beginning of the production process. It may belong either to a discrete set $\Omega = \{s_0, \dots, s_k\}$ of possible durations, or simply be bounded by a maximum value S . The disturbance neighborhood is then the real interval $[0, S]$.

4.2.2.2. Sensitivity of the *HM* algorithm

The solutions provided by *HM* are extremely sensitive to disturbances. Let us consider the example of n tasks of unit duration, and deadlines $d_j = j$. The optimal solution schedules all tasks exactly in time. Imagine now that the machine is unavailable during the first time slot. All tasks are now late. The optimal solution for $s = 1$ consists of disqualifying the first task. The sequence obtained from *HM* is then at distance $n - 1$ from the optimal for the considered robustness measure.

From this example, it is apparent that considering the solutions which are optimal in the total availability case is not very wise. Furthermore it is not possible to bound the performance loss obtained by *HM*, with respect to the disturbance size, and thus find a sensitivity bound of *HM* solutions. The only interesting robustness criterion (if stochastic models are left aside) is therefore minimizing the worst case absolute robustness (that is, minimizing the maximum regret).

4.2.2.3. Hypotheses and notations

Let us suppose that without disturbance, all n tasks can be scheduled without lateness. The study focuses on the case where a sequence is fixed: it is the sequence obtained from *HM* without disturbance, noted σ_0 . The objective is to find the set of tasks that disqualify σ_0 from obtaining a robust solution. Symmetrically, the aim is to

find the best sequence σ_R from σ_0 . Tasks are numbered in σ_0 order. Disqualified tasks are considered as late for all scenarios, hence they can be considered as non-scheduled tasks. To justify these two hypotheses, let us consider again a firm that at the beginning of each period receives a set of commands. Using the *HM* algorithm, it computes a first schedule and refuses the tasks that are late for this schedule, thus minimizing the number of tasks to reject. In order to attenuate the effects of a possible disturbance, the firm can choose to disqualify some more commands, which are then considered as lost.

For a given s , denote by $f(K, s)$ the number of late tasks in the sequence obtained by disqualifying the set K , and $opt(s)$ the optimal number of late tasks for scenario s . This optimum is simply obtained by applying *HM* on deadlines $d'_j = d_j - s$. The objective is then to find a set of tasks K_R which minimizes the metric:

$$R_{DEV}(K) = \max_{s \in [0, S]} \{f(K, s) + |K| - opt(s)\}$$

The task margin is the difference between its deadline and its finishing time: $\text{Margin}(j) = d_j - C_j$, for *HM* without disturbance. In the specific case where task margins are increasing in sequence σ_0 obtained by *HM*: $i < j \Rightarrow \text{Margin}(i) \leq \text{Margin}(j)$, the robust solution has a specific structure that is exploited below. For more details see [MAH 04].

4.2.2.4. The two scenario case

Let us suppose that only two scenarios are possible: $\Omega = \{0, S\}$. If $s = 0$, the optimal solution consists of disqualifying zero tasks. If $s = S$, the $opt(S)$ first tasks are disqualified. Imagine now that the set $K_R = K_{\{0\}}^*$, S of tasks disqualified by the robust optimal solution is known. In this sequence denote by x_S the first task that is in time for disturbance S . A task placed after x_S in initial sequence σ_0 cannot be in K_R . Indeed, such a task has a larger margin than x_S by property of σ_0 . Hence it is on time if x_S is; disqualifying it would strictly diminish the number of tasks in time for both scenarios. For x_S to respect its deadline for disturbance S , disqualified tasks should satisfy:

$$\sum_{j \in K_R} p_j \geq S - \text{Margin}(x_S) \quad (4.1)$$

As the distance between a robust solution and the optimal solution for disturbance $s = 0$ is the number of disqualified tasks $|K_R|$, among the sets K of tasks verifying

equation (4.1), K_R has minimal cardinality. Once task x_S is known, finding K_R is equivalent to find the task set K verifying:

$$K_R(x_S) = \arg \min |K| \text{ such that } \sum_{j \in K} p_j \geq S - \text{Margin}(x_S) \quad (4.2)$$

Hence it is sufficient to disqualify the largest tasks before x_S , x_S excluded, until a positive margin is obtained for x_S . With a structure that maintains the list of tasks before x_S sorted, determining $\mathcal{K}_R(x_S)$ is done in time $\mathcal{C}(n)$. This solution quality is then

$$\max \{|K_R(x_S)|, x_S - 1 - \text{opt}(S)\}.$$

As n tasks are candidates to be x_S , determining robust optimal solution can be done in time $\mathcal{C}(n^2)$.

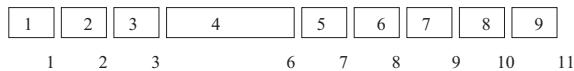


Figure 4.1. Initial sequence

Let us consider the sequence of Figure 4.1, where due dates are equal to completion times in the initial solution. The possible disturbance S is of duration 3. For this sequence, if no task is disqualified, the maximum deviation is 6 and it is obtained for $S = 3$. However, if tasks 1, 2 and 3 are disqualified (these are the tasks disqualified in the optimal solution for $S = 3$) the maximum deviation is 3, obtained for $s = 0$. The robust-optimal solution consists of disqualifying task 4 alone. The maximum deviation is then of 1, and it is reached in both scenarios. For this example, task x_S is task 5, hence in order to keep task 5 on time (and all tasks scheduled after it) it is sufficient to disqualify task 4, the largest task scheduled before 5, in order to verify equation (4.2).

Finding a robust-optimal solution in polynomial time is also possible in the case of an arbitrary number of scenarios. It is done by relying on the continuous case (see theorem 4.1). The algorithm is based upon dynamic programming; it looks for the set of tasks to disqualify from the initial sequence. For more details the interested reader can refer to [MAH 04].

THEOREM 4.1.– *The robust-optimal solution for absolute deviation measure can be found in $\mathcal{C}(n^4)$ for problem $1||\sum U_j$, on any disturbance interval like $[0, S]$, when the margins of the tasks are increasing in initial sequence σ_0 .*

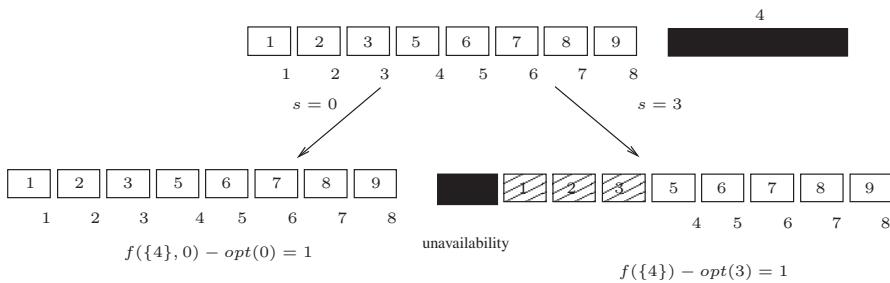


Figure 4.2. Robust sequence

Concerning a sequence which does not verify the non-decreasing margin hypothesis, the robust subsequence may be found by using dynamic programming, and scanning the set of possible beginning times for the tasks. The algorithm is pseudo-polynomial. In general, when the sequence is not fixed, the problem is still open.

4.3. m -machine problems without communication delays

4.3.1. Parametric analysis

This part presents the few works proposing a parametric analysis, that is to say, examining the effect of one disturbance on one parameter. For Hall and Posner, as for Sotskov, a solution or a method quality is measured by the extent of the disturbances for which this solution or method remains optimal (or keeps a fixed performance guarantee); hence robustness is assimilated to stability. In case of optimality loss, re-optimization is implicitly considered as always possible, and the complexity of this new problem (fourth question) is considered.

Picouleau has more recently completed the results from Hall and Posner [HAL 04] by showing that in many cases, modifying one parameter by one unit only was sufficient to make the re-optimization \mathcal{NP} -complete. Re-optimization complexity is the subject of Chapter 14 of this book. Unfortunately, re-optimization is not always possible. First, the exact values of some parameters might remain unknown at some date t . Second, at some date t , it may be impossible to recompute completely the schedule: either for stability reasons or because of delay constraints. It seems to us that this occurs frequently in scheduling applications.

Hall and Posner did look for performance relative deviation of an optimal solution in some specific cases. For problem $P2 \parallel \sum w_j C_j$, they proved the existence

of a deviation depending on the disturbance size on an execution time, and on initial sequence of the jobs. The results hold for some additional conditions on the disturbance. Their study is particularly interesting for problem $P2\|C_{\max}$; it is first shown that if disturbance Δ does not exceed some threshold, the relative deviation is bounded by $8/7$. The threshold may be computed in $\mathcal{O}(n \log n)$ but depends on S^* . If the threshold is exceeded, S^* cannot be optimal anymore and the bound does not hold. But if a slight modification is made to S^* , the obtained solution S' is no longer optimal on initial data; however, the ratio $8/7$ is still verified for a much larger set of disturbances (for instance, if in S^* both machines finish at the same time, the initial threshold is multiplied by $10/3$, which is the maximum multiplier). Note that the threshold for S^* constitutes a robustness measure for S^* . S' is more robust than S^* in the sense that its approximated stability ratio (for value $8/7$, and a unique parameter) is larger.

Let us consider the practical use of these results. If the parameter subject to disturbances is known (here, an execution duration), and if the disturbance magnitude may be foreseen to some extent, then it might be interesting to slightly modify the initial optimal schedule to obtain a suboptimal, but more robust solution. It would be interesting to check for other problems, where similar results hold, particularly for m machines.

When the deterministic problem is polynomial, Hall and Posner show that an *a priori* analysis allows us to diminish re-optimization time, as previously seen for one machine. The complexity of this computation depends on the number of modified parameters, but also on the deterministic-optimal schedule and on the position of the tasks concerned by the changes. Optimal algorithms for studied problems are list algorithms, priority algorithms or the simplex. Hall and Posner show clearly in this last case that a parametric analysis of the linear program cannot be sufficient. Indeed, modifying one parameter of the scheduling problem may entail the modification of several parameters of the LP. Note, however, that if the SA can be executed off-line, its complexity may not be a major issue.

Hall and Posner also proposed several proactive methods for building robust solutions, as for $1\|\sum w_j U_j$ (see section 4.2.1). In the case of $Pm|pmtn|C_{\max}$, McNaughton's simple algorithm can build a large number of optimal solutions. If the task T_j whose duration might change is known, it is advisable to spread this task on the maximum number of machines, thus sharing the additional load among these machines (it is the same if the task duration decreases). The authors present a linear algorithm to achieve this. The obtained solution is not McNaughton's, and

may have more preemptions. However, it remains optimal whatever the disturbance on p_j . This result should be discussed. First, if T_j is unknown, the authors do not investigate a solution distributing all tasks at best. We may consider, however, that in case of a processing time increasing, it will be distributed on all machines, even on those that did not execute T_j . The diminishing case is more complex. Second, the proposed method consists of increasing or decreasing each piece of T_j . This supposes we fact to know the disturbance before execution of T_j begins (more an off-line re-optimization than an on-line repair).

Kolen *et al.* [KOL 94] study different list methods for independent tasks and makespan minimization. Priorities depend on processing times (for instance SPT, LPT, etc.). They observe the number of possible schedules obtained by each method when the processing time of a task varies. In the framework used in this book, they measure the flexibility of these methods (see Chapters 10 and 11). They also study the obtained makespans. There is no surprise: methods building best deterministic schedules are also less flexible. Flexibility is assimilated in the article to sensitivity, in the sense that a less flexible method will be more sensitive to disturbances. This work might be applied in a proactive/reactive scheme: the study allows us to choose the list method that will be used on-line. In other words, the schedule is not built before the execution, but the on-line policy, that is, the set of rules to apply according to different conditions, is fixed. With different tools, the stochastic scheduling approach is similar: a policy is tested *a priori*, according to criteria (like makespan expectation) taking uncertainties into account.

4.3.2. Example of global analysis: $Pm \parallel \sum C_j$

Question 3 is considered in this section, where already we seek to determine the performance loss caused by disturbance. As was shown, classical analysis cannot answer unless restrictive hypotheses are made. Often, one unique parameter is modified; the magnitude of the disturbance is limited, and furthermore this limitation depends on the studied solution itself; it is not a choice of the user. Last, robustness measures are in fact stability measures, which are less informative than other measures. Here the worst case relative deviation is used. Gerasoulis and Yang [GER 95], then Penz, Rapine and Trystram [PEN 01] propose to use as robustness criterion a function s of the disturbance magnitude, called *sensitivity guarantee*. For a given off-line algorithm \mathcal{A} , $s_{\mathcal{A}}(\varepsilon)$ is equal to the worst case relative deviation when the disturbance magnitude is at most ε with regard to an estimated instance, divided by the performance ratio of \mathcal{A} in the deterministic case. This last ratio is of course equal to 1 when \mathcal{A} is optimal. Otherwise, the sensitivity guarantee expresses the increase in the performance loss due to uncertainties. An algorithm may be

considered as robust if it admits a sensitivity at most linear in ε (a less strong demand would be that $s_{\mathcal{A}}(\varepsilon)$ is polynomial in ε). Here it is the sensitivity of one algorithm that is analyzed, the most robust solution is not investigated.

Hence, from a fixed processing time vector \tilde{p} , a neighborhood of \tilde{p} is considered, whose size is given by ε . A key feature of this approach is to give a correct definition of the magnitude. The definition of [GER 95, PEN 01] is the following for processing times. Let \tilde{p}_j be the estimated duration for task T_j , and $p_j = (1 + \epsilon_j)\tilde{p}_j$ be the effective duration.

DEFINITION 4.3.— *The disturbance magnitude $\vec{\varepsilon} = (\varepsilon_1, \dots, \varepsilon_n)$ is the real ε defined by:*

$$1 + \varepsilon = \frac{1 + \varepsilon^+}{1 - \varepsilon^-}$$

with $\varepsilon^+ = \max_{i=1,n}^{\oplus}\{\varepsilon_i\}$ and $\varepsilon^- = \max_{i=1,n}^{\oplus}\{-\varepsilon_i\}$. The function \max^{\oplus} is defined as $\max^{\oplus} A = \max A \cup \{0\}$. A disturbance vector $\vec{\varepsilon}$ is an ε -disturbance if its magnitude is less than or equal to ε .

This definition is rather precise. It corresponds to the intuitive idea of comparing the largest and smallest ratios between real and estimated durations $\frac{p_j}{\tilde{p}_j}$.

THEOREM 4.2.— [PEN 01] *Let us consider a scheduling problem with tasks bound by precedence constraints, and arbitrary resources constraints (but with constant amount of resources). The criterion is either the makespan minimization or the flow time minimization. Let \mathcal{A} be an off-line algorithm. Then*

$$s_{\mathcal{A}}(\varepsilon) \leq (1 + \varepsilon);$$

Thus, the sensitivity guarantee is at most linear in the disturbance magnitude, and this for a very large family of scheduling problems. This result is obtained through rather large overestimates, and for any algorithm. We might hope to do better in many cases. Furthermore, when \mathcal{A} is optimal, an upper bound of the worst case relative deviation is obtained. If \mathcal{A} is not optimal, to obtain such an upper bound we must multiply s by its performance ratio (see above the definition of $s_{\mathcal{A}}$). If this ratio is bounded by a constant, the optimization remains linear in the disturbance magnitude. This result was first proposed by [GER 95] in the case of an unbounded number of processors and C_{\max} , and it is reached by the DSC algorithm. Penz *et al.* [PEN 01] show it is asymptotically reached for m machines and makespan minimization. The worst case arises when independent tasks have the same estimated processing times.

Penz *et al.* show it is possible to do better in the case of m machines, independent tasks and flow time minimization. The list algorithm based upon the SPT rule is optimal.

THEOREM 4.3.– *The sensitivity guarantee $s_A(\varepsilon)$ of SPT for $1\parallel \sum C_i$, and for $Pm\parallel \sum C_i$, is bounded by $\sqrt{1+\varepsilon}$.*

This bound is asymptotically reached once again for equal processing times.

The result shows that SPT, which is optimal on the estimated instance, is not very sensitive to disturbances: if, for instance, the task duration is allowed to double in the real instance with respect to estimated processing times, the performance degradation of the solution remains bounded by $\sqrt{2}$. Remember that this is different from the real increase of the flow time (absolute stability measure) which can be larger (in the example the flow time can be multiplied by 2).

Unfortunately, this bound in the square root of the magnitude might be difficult to obtain for other problems, particularly for other criteria than the flow time which is an additive criterion. There are still few results of that type, which are yet rather significant. Of course the definition of the magnitude has a great influence on the functions s . Conversely, it is a great temptation to define the magnitude according to the results which have already been found (see also section 4.4.3 for some additional comments on the subject)!

4.4. The m -machine problems with communication delays

In this section, the model derived from Rayward-Smith [RAY 87] and Papadimitriou and Yanakakis [PAP 90] is studied: a set of tasks bound by precedence constraints must be executed on identical machines. The tasks have different processing times. Furthermore, if two tasks bound by a precedence constraint are executed by two different processors, there is a delay, called a communication delay, between the end of the first and the beginning of the second task. Overlapping is allowed: one machine may communicate (send or receive data) and process some task simultaneously. However, neither preemption nor duplication are allowed. Finally, the performance criterion is the makespan.

Inside a workshop, the communication delay models the transportation time of one piece from one machine to another. In parallel computing, it models the time to send a message, like the result of one first computation used as an entry for another executed on another processor. There is a large literature on workshop problems with

transportation times, usually not with a parallel model; it will not be explicitly studied here.

In parallel computing, difficulties are of two kinds: first, it is difficult to divide some applications into tasks small enough to be able to evaluate their processing times; second, the interconnection network between the processors or machines of the parallel system has to be taken into account. Indeed, even if the network is not considered as a critical resource, for which it should be necessary to know the precise schedule of the communication tasks (thus necessitating a much more complex model), foreseeing the duration of some communication is not easy. It depends on the message length, on the source and destination of the message, and on the path chosen to route the message. The *routing* problem itself is complex. According to the choices made in the target architecture, the computation of the communication times is very different (traditional communication modes are *store and forward*, *circuit switching*, *worm-hole*, etc.). Even for a fixed architecture, it is difficult to precisely compute this time because of communication link availability or network contention. Other models were proposed, like divisible tasks, where a task may be executed simultaneously on different machines (see [BŁA 00]). They are outside the scope of this study.

The optimality results for the model are often obtained with the hypothesis that communication delays are identical, whatever the tasks and their allocation may be. However, for new problems (Internet use, grid computing) a deterministic model for the communication delays does not seem to be adapted. Thus, looking for robust schedules with respect to some uncertainty on these delays is a very promising trail. However, there are still few works on the subject. Here a complete sensitivity analysis is presented in the two machine case, with unitary estimated processing times, and trees; then bounds are proposed in a more general framework. They are applied in the case of an unbounded number of processors. These analyses consider uncertainties on communication delays only, processing times being fixed. A proactive/reactive approach for this model becomes interesting when SA shows the inadequacy of a pure proactive approach. See [MOU 99, MOU 03, GUI 04, SAN 07] and Chapter 12.

4.4.1. Notations and definitions

In most models (and specifically for that used here) the communication delay depends on initial and final tasks, T_i and T_j respectively, and not on the machines that execute them. It is noted c_{ij} . The number of the machine assigned to task T_j by schedule S is $\pi_j(S)$. When execution durations are unitary and communication costs are zero, the problem is called UET. When communication delays are unitary too, it is

called UECT. If communication delays are less than equal to (respectively larger than or equal to) execution durations, the problem is called SCT, or with coarse granularity (respectively LCT or with small granularity).

DEFINITION 4.4.— *A communication is effective between two tasks T_i and T_j for schedule S if and only if $T_i \prec T_j$ or $T_j \prec T_i$ and $\pi_i(S) \neq \pi_j(S)$.*

The communication delay is noted \tilde{c}_{ij} for estimated value and c_{ij} for effective value during the execution phase. Furthermore, \bar{c} (resp. \underline{c}) denotes the maximum (resp. minimum) delay of an effective communication.

A complete schedule – assignment of tasks to processors, execution order on each processor, and beginning times of each task – is built before execution phase using estimated delays. The schedule obtained with the real costs admits the following properties:

- assignment is not modified,
- the execution order of the tasks on each processor is not modified,
- new execution times of the tasks are obtained by executing each task as early as possible.

The sensitivity analysis in this section consists of studying the differences between the beginning times in initial schedule \tilde{S} obtained by a given algorithm and the final schedule S . Let $\tilde{\omega}$ be the makespan of \tilde{S} , and ω be the makespan of S . The optimal makespan is noted ω^* for the effective communication delays. The notation ω_c will sometimes be used for a cost vector c , with ω_0 the makespan for zero delays, and ω_1 the makespan for unitary delays. By a slight abuse of notation, $c \in [\underline{c}, \bar{c}]$ means that all coordinates of vector c are inside this interval. With these notations, the at worst relative (resp. absolute) deviation of S is $\max_{c \in [\underline{c}, \bar{c}]} \frac{\omega}{\omega^*}$ (resp. $\max_{c \in [\underline{c}, \bar{c}]} \omega - \omega^*$).

DEFINITION 4.5.— *A sensitivity relative (resp. absolute) bound of an algorithm is an upper bound of the ratio $\frac{\omega}{\omega^*}$ (resp. of the difference $\omega - \omega^*$) over all instances of the problem.*

Such bounds are expressed as functions of \bar{c} and \underline{c} . They are upper bounds of relative and absolute worst case deviations, when effective communication delays are within interval $[\underline{c}, \bar{c}]$, and will be used as robustness measures for the considered algorithms.

The list algorithms might be extended to problems with communications in several ways. The most natural extension, and the most frequently used one, shall be considered here (see Hwang *et al.* [HWA 89], Hanen and Munier [HAN 98]). At time t , a task T_k is ready if its execution can begin on at least one machine at time t . Because of communications, ready tasks are of two kinds. If each predecessor T_j of T_k is finished before $t - c_{jk}$, T_k might be executed at t on any machine and is called *free*. In the other case, there is a predecessor T_j that is finished after $t - c_{jk}$, and T_j cannot be executed at t , except on the same machine. It is ready, but *bound* to this machine. It follows that only the ready task bound to a machine M_i with highest priority can be executed on M_i . [HWA 89] calls this extension ETF (earliest task first). A significant part of the literature is devoted to the performance of these methods.

For instance, Moukrim *et al.* [MOU 03] studied the *fork* and *join* graphs with uncertainties. A fork graph contains one task that precedes all the others (which are two by two independent). A join graph is a reversed fork graph. In the static case ETF is optimal for these graphs, whatever the duration times and communication delays might be. However, it is shown quite simply that the stability radius of ETF is zero in both cases. Such negative results on the stability radius, even for very simple graphs, explains why studies focus on deviation measures.

4.4.2. The two-machine case

The complexity of $P2|prec, UECT|C_{\max}$ is unknown. The list algorithm ETF (without any particular order on the ready tasks) has a bound of 2. The ETF-CG algorithm (Coffman-Graham order) has a bound of $\min(3/2, 4/3 + 3/w^*)$ that is tight.

The $P2|tree, UECT|C_{\max}$ problem is polynomial and we now present a sensitivity analysis of the algorithms solving this problem. The detailed analysis may be found in [GUI 04].

DEFINITION 4.6.– A processor-ordered *assignment* is an assignment such that all effective communications are from one processor, noted P_s (sender), toward the other one, noted P_r (receiver). A processor-ordered schedule (PO) is such that its assignment is processor-ordered.

In what follows, PO-schedules denote the processor-ordered schedules.

Suppose that all communication delays are zero, and let us consider the schedules obtained by Hu's algorithm (list schedules with level by level priority) that is optimal

for UET trees. The resulting schedules verify that P_r is without idleness until the end of the schedule, and P_s is without idleness until some time t ; after t , it does not execute any other task. By using a task exchange argument, the following result can be shown:

THEOREM 4.4.— *For any instance of $P2|\text{tree, uet}|C_{\max}$, there exists one PO-schedule that is optimal.*

Then a bound on the difference between the optimal makespan with and without communication delays can be derived:

THEOREM 4.5.— *The difference between the optimal makespan with and without communication delays is bounded by \bar{c} : $\omega^* - \omega_0^* \leq \bar{c}$.*

Because an optimal PO-schedule exists for effective communication costs, it is easy to verify that $1 - \underline{c}$ is an absolute sensitivity bound. Unfortunately, we proved that the PO-schedules are not dominant in general for SCT problems. It is however possible to show the existence of the same bound for any UECT-optimal schedule; see [GUI 04] for the proof details.

THEOREM 4.6.— *Optimal schedules for UECT trees admit an absolute sensitivity bound in the SCT case of at most $1 - \underline{c}$:*

$$\omega - \omega^* \leq 1 - \underline{c}$$

For LCT (large communication delays) problems, a relative sensitivity bound has been established:

THEOREM 4.7.— *Optimal schedules for UECT intrees admit a relative sensitivity bound in the LCT case equal to $\frac{\bar{c}+1}{2}$.*

We have also established a general result for PO-schedules:

THEOREM 4.8.— *Let us consider a problem with $\bar{c} \geq 1 \geq \underline{c}$. The optimal PO-schedules for UECT intrees admit an absolute sensitivity bound of $\bar{c} - \underline{c}$.*

Four algorithms have been proposed to compute optimal schedules for UECT trees on two processors [PIC 92, VEL 93, LAW 93, GUI 00]. None of the first three systematically build PO-schedules, and the relative sensitivity bound $\frac{\bar{c}+1}{2}$ (see theorem 4.7) is tight for the three algorithms. By contrast, the schedules obtained by the fourth algorithm of [GUI 00] (denoted CBOS: Clusters Based on Subtrees) are PO. Hence for problem $P2|\text{tree, uect}|C_{\max}$, there is always an optimal PO-schedule.

The results given above on PO-schedules imply that CBOs admits an absolute sensitivity bound of $\bar{c} - \underline{c}$. This algorithm assigns to the second processor P_s some task groups forming intrees (if a task is on P_s , all of its predecessors are too), and respects a balance property between both processors. From each subtree of P_s , there is one and only one communication towards P_r .

This example highlights the great difference in terms of robustness between two algorithms, which are both optimal in the static case. It is logical for processor ordered schedules to obtain better performances, as they limit the number and the impact of communications. One perspective is the extension and study of such algorithms for m machine problems. This extension can be achieved in different ways, including the convex clustering presented and studied in a proactive/reactive framework in Chapter 12.

4.4.3. The m -machine case

First, the performances of some algorithms are presented without uncertainty. Specific sensitivity analysis might be performed on these algorithms, but the general results presented afterward are applied to them instead.

4.4.3.1. Some results in a deterministic setting

The main results are given in [CHR 95]; see also [HAN 98, VAR 96]. Outside the two processor case, there are few polynomial problems. Let us consider first the case of an infinite number of machines (an unbounded number of machines is more appropriate).

Without communication, this is the central scheduling problem, often called (somewhat abusively) the PERT problem. Thus, it is sufficient to compute the critical path for finding a minimal makespan schedule. Adding communications changes things. Indeed, with any problem with communication delays, computing the length of a path cannot be done before assignment is made (as this length takes into account the effective communications). The problem is NP-hard for tree precedence graphs and arbitrary communication delays.

Chrétienne [CHR 89] proposes a polynomial algorithm in the case of SCT trees. The algorithm is adapted from ETF for this particular case. The resulting schedule is linear (two independent tasks are not executed on the same machine), as with algorithms based on clusters. Without SCT hypothesis, list schedules are no longer dominant, due to non-dominance of linear schedules: it might be necessary to delay some task to execute it on the same processor as its successor.

Many approximation results exist (positive and negative) in the case of m machines and usually in UECT hypothesis; see [CHR 95, HAN 01]. The algorithms are based on ETF list methods or on clusters, such as DSC algorithms of [GER 92] and the CBoS algorithm from the previous section that can be generalized to m machines. It then admits (see [GUI 97]) an absolute performance bound $\omega - \omega^* \leq \frac{m-1}{2}$. Cluster-based algorithms have the effect of minimizing the number of effective communications, hence they are interesting methods to diminish the uncertainty impact, as shown by the two-machine case.

4.4.3.2. Framework for sensitivity analysis

In this section a sensitivity analysis is presented concerning the communication delays. The precedence graph is arbitrary. For any precedence $T_i \prec T_j$ represented by the arc a_k , an estimated delay \tilde{c}_k and an interval $[\alpha_k, \beta_k]$ are given. By hypothesis, the communication delays (real and estimated) are inside this interval. Denote by α and β the n -vectors composed of the lower and upper bounds of these intervals. In the most general case, these quantities might depend on the task assignment, hence the studied schedule, see [SAN 05]. A worst case analysis is made, and sensitivity bounds are computed for given vectors α and β . The above notations can be used: $\bar{c} = \max \beta_k$ and $\underline{c} = \min \alpha_k$. The execution durations are fixed, and we use notation $p = \min p_j$. Last, by reference to the study of section 4.3.2, a possible definition of the perturbation magnitude is here described as $\varepsilon = \bar{c} - \underline{c}$.

4.4.3.3. Stability studies

Let us first consider the stability problem, that is the performance loss of some schedule S after some disturbance, see Chapter 1. This loss is measured by the stability ratio (relative stability) $st_{\tilde{c}}(S) = \frac{\omega_c(S)}{\omega_{\tilde{c}}(S)}$, and when it does exist by the stability difference or absolute difference (absolute stability) $ast_{\tilde{c}}(S) = \omega_c(S) - \omega_{\tilde{c}}(S)$.

As might be expected, no upper bound of $ast_{\tilde{c}}(S)$ independent of the graph size exists, and this remains true in really special cases. [MOU 03] considers for instance the unbounded case and Chrétienne's algorithm.

THEOREM 4.9.— Consider the problem $P\infty|tree, SCT|C_{\max}$. Let h be the height of the task graph (considering the arcs), l its width, and $\varepsilon = \bar{c} - \underline{c}$. Let S be a schedule computed by Chrétienne's algorithm for delays \tilde{c} . Then

$$ast_{\tilde{c}}(S) \leq \min(h, l - 1) \times \varepsilon$$

and this bound is tight.

From the theorem, there indeed exists an absolute stability bound, but for either height bounded or width bounded graphs. This result is obtained using the hypotheses of unboundedness on m and of linearity of the obtained schedule.

However, it is possible to bound the stability ratio. Let us first note that inside the chosen framework, $st_{\tilde{c}}(S) = \frac{\omega_{\beta}(S)}{\omega_{\tilde{c}}(S)}$, as the makespan naturally increases with the delays for a fixed S .

Note. Let x and y be two communication vectors.

$$W(x, y) = \frac{1 + \frac{p}{\max_k y_k}}{\min_k \frac{x_k}{y_k} + \frac{p}{\max_k y_k}}$$

THEOREM 4.10.– [SAN 05] *For any estimated delay vector \tilde{c} ,*

$$st_{\tilde{c}}(S) \leq W(\tilde{c}, \beta)$$

In order to lighten the notations, the fact that the quantities W can in fact depend on S is omitted. The expression is much simplified if equal intervals are considered: $[\underline{c}, \bar{c}]$ for all effective communications. The best bound is obtained in the case of equal estimations $\tilde{c}_k = u \geq \underline{c} \forall k$. In that case,

$$st_u(S) \leq \frac{p + \bar{c}}{p + u}.$$

4.4.3.4. Sensitivity bounds

Suppose now that an optimal schedule S is available for the estimated delays \tilde{c} . Because of the above observations on the stability difference, it seems very difficult to bound the worst case absolute deviation of S , as was done in section 4.4.2, and bounds on the relative deviation are sought instead. A general bound is proved in [SAN 05]. From it, we can deduce:

THEOREM 4.11.– *Suppose that $\forall k, c_k \in [\alpha_k, \beta_k]$. If S is optimal for delays α , or if S is optimal for delays β , then*

$$W(\alpha, \beta) = \frac{p + \max_k \beta_k}{p + \max_k \beta_k \cdot \min_k \frac{\alpha_k}{\beta_k}}$$

is a relative sensitivity bound for S .

If S is an optimal schedule for an estimated vector different from α and β , the bound is less interesting (for equal vectors α and β). As for stability, the expression of the bound is much simpler in the equal interval case:

THEOREM 4.12. – Suppose that $\forall k, c_k \in [\underline{c}, \bar{c}]$. If S is optimal for constant delays $u \in [\underline{c}, \bar{c}]$, then

$$\frac{p + \bar{c}}{p + \underline{c}}$$

is a relative sensitivity bound for S .

The bound of theorem 4.12 is tight in two particular cases. Let us first consider the problem studied in section 4.4.2; then $p = 1$, and in the LCT case ($\underline{c} = 1$) any UECT-optimal schedule admits a relative bound of $\frac{1+\bar{c}}{2}$ (theorem 4.7). Remember that the CBoS algorithm admits an absolute bound because it is processor-ordered, but theorem 4.12 does not assume any hypothesis on S , apart from its optimality in the deterministic case.

Let us now consider the problem for which Chrétienne's algorithm is optimal, $P\infty|tree, sct|C_{max}$. Any vector \tilde{c} can be chosen for the estimated delays, while the algorithm builds an optimal schedule for these delays as far as the SCT hypothesis is respected: $\bar{c} \leq p$. In particular, theorem 4.12 applies. Furthermore it is easy in that case to show that the bound is tight (for instance, when complete binary trees are considered). This bound is independent of the estimation used to build the schedule.

Let us finally note that this bound, contrary to section 4.3.2, cannot be expressed as a function of a unique parameter describing the size of the disturbance or of the uncertainty; both parameters \underline{c} and \bar{c} are necessary. Another difference comes from the nature of the data subject to uncertainty. If it is a processing time as in section 4.3.2, this duration is at least known *a posteriori*. However, if a communication is not effective in the chosen schedule, the associated real delay will never be known. Hence *a posteriori* analysis is questionable (and can not help to verify the adequacy of the hypotheses on the delays).

4.5. Conclusion

The objective of this chapter was to present an overview of sensitivity analysis applied to single and m machine problems. In order to do that, it has been necessary to define what is, or what should be according to us, such analysis, its interest and its limits. A number of recent results have been presented. The analyses were sometimes

completed by the search for solutions maximizing robustness, which is in general a more difficult problem than the analysis of a specific algorithm. Within the scope of this book, a sensitivity analysis and/or the search for robust solutions should help to define the theoretical performances of a predictive or proactive approach, before eventually proposing a fully proactive/reactive approach, which always more difficult to elaborate and to use.

4.6. Bibliography

- [BEN 99] BEN-TAL A. and NEMIROVSKI A., “Robust solutions of uncertain linear programs”, *Operations Research Letters*, vol. 25, p. 1–13, 1999.
- [BER 03] BERTSIMAS D. and SIM M., “Robust discrete optimization and network flows”, *Mathematical Programming series B*, vol. 98, p. 49–71, 2003.
- [BŁA 93] BŁAŻEWICZ J., ECKER K., SCHMIDT G. and WEGLARZ J., *Scheduling in Computer and Manufacturing Systems*, Springer-Verlag, 1993.
- [BŁA 00] BŁAŻEWICZ J., “Scheduling for computer science”, *Colloque : Ordonnancement Déterministe pour l’Informatique Parallèle*, Aussois, France, September 2000.
- [CHA 04] CHANAS S. and KASPERSKI A., “Sensitivity analysis in the single-machine scheduling problem with min-max criterion”, *Int. Trans. in Operational Research*, vol. 12, p. 287–298, 2004.
- [CHR 89] CHRÉTIENNE P., “A polynomial algorithm to optimally schedule tasks on a virtual distributed system under tree-like precedence constraints”, *European Journal of Operational Research*, vol. 43, p. 225–230, 1989.
- [CHR 95] CHRÉTIENNE P. and PICOULEAU C., “Scheduling with communication delays: a survey”, p. 65–90, in P. Chrétienne et al., *Scheduling Theory and its Applications*, John Wiley, 1995.
- [DAN 95] DANIELS R.L. and KOUVELIS P., “Robust scheduling to hedge against processing time uncertainty in single stage production”, *Management Science*, vol. 41, no. 2, p. 363–376, 1995.
- [GAB 07] GABEL V., MURAT C. and REMLI N., Linear programming with linear right hand sides, Report no. 255, Cahiers du Lamsade, Paris-Dauphine University, France, June 2007.
- [GAL 97] GAL T. and GREENBERG H.J. (Eds.), *Advances in Sensitivity Analysis and Parametric Programming*, Kluwer Academic Publishers, Boston, 1997.
- [GER 92] GERASOULIS A. and YANG T., “A comparison of clustering heuristics for scheduling DAGs on multiprocessors”, *J. of Parallel and Distributed Computing*, vol. 16, p. 276–291, 1992.

- [GER 95] GERASOULIS A. and YANG T., “Applications of graph scheduling techniques in parallelizing irregular scientific computations”, p. 245–267, in FERREIRA A. and ROLIM J. (Eds.) *Parallel Algorithms for Irregular Problems: State-of-the-Art*, Kluwer, 1995.
- [GUI 97] GUINAND F., RAPINE C. and TRYSTRAM D., “Worst case analysis of algorithms for scheduling UECT trees on m processors”, *IEEE Trans. on Par. and Distr. Systems*, vol. 8, 1997.
- [GUI 00] GUINAND F. and TRYSTRAM D., “Optimal scheduling of UECT trees on two processors”, *RAIRO Operations Research*, vol. 34, p. 131–144, 2000.
- [GUI 04] GUINAND F., MOUKRIM A. and SANLAVILLE E., “Sensitivity analysis of tree scheduling on two machines with communication delays”, *Parallel Computing*, vol. 30, p. 103–120, 2004.
- [HAL 04] HALL N.G. and POSNER M.E., “Sensitivity analysis for scheduling problems”, *Journal of Scheduling*, vol. 7, p. 49–83, 2004.
- [HAN 98] HANEN C. and MUNIER A., “Performance of Coffman Graham schedule in the presence of unit communication delays”, *Discrete Applied Mathematics*, vol. 81, p. 93–108, 1998.
- [HAN 01] HANEN C. and MUNIER A., “An approximation algorithm for scheduling dependent tasks on m processors with small communication delays”, *Discr. Appl. Math.*, vol. 108, no. 3, p. 239–257, 2001.
- [HWA 89] HWANG J.J., CHOW Y.C., ANGER F.D. and LEE C.Y., “Scheduling precedence graphs in systems with interprocessor communication times”, *SIAM Journal on Computing*, vol. 18, no. 2, p. 244–257, 1989.
- [KAS 05] KASPERSKI A., “Minimizing maximal regret in the single machine sequencing problem with maximum lateness criterion”, *Operations Research Letters*, vol. 33, no. 4, p. 431–436, 2005.
- [KOL 94] KOLEN A.W.J., RINNOY-KAN A.H.G., HOESEL C.P.M.V. and WAGELMANS A.P.M., “Sensitivity analysis of list scheduling heuristics”, *Discrete Applied Mathematics*, vol. 55, no. 2, p. 145–162, 1994.
- [LAW 93] LAWLER E.L., “Scheduling trees on multiprocessors with unit communication delays”, *Workshop on Models and Algorithms for Planning and Scheduling Problems*, Villa Vigoni, Lake Como, Italy, 1993.
- [LEB 06] LEBEDEV V. and AVERBAKH I., “Complexity of minimizing the total flow time with interval data and minmax regret criterion”, *Discrete Applied Mathematics*, vol. 154, no. 15, p. 2167–2177, 2006.
- [MAH 04] MAHJOUR A., Etude de la Robustesse et de la Flexibilité pour des problèmes d’ordonnancement et de localisation, PhD thesis, INPG, Grenoble, France, 6 July 2004.

- [MOU 99] MOUKRIM A., SANLAVILLE E. and GUINAND R., “Scheduling with communication delays and on-line disturbances”, in AMESTOY P. (Ed.), *Euro-Par'99*, Toulouse, France, LNCS 1685, p. 350–357, 1999.
- [MOU 03] MOUKRIM A., SANLAVILLE E. and GUINAND R., “Parallel machine scheduling with uncertain communication delays”, *RAIRO Operations Research*, vol. 37, p. 1–16, 2003.
- [PAP 90] PAPADIMITRIOU C.H. and YANNAKAKIS M., “Towards an architecture-independent analysis of parallel algorithms”, *SIAM Journal on Computing*, vol. 19, no. 2, p. 322–328, 1990.
- [PEN 01] PENZ B., RAPINE C. and TRYSTRAM D., “Sensitivity analysis of scheduling algorithms”, *European Journal of Operational Research*, vol. 134, p. 606–615, 2001.
- [PIC 92] PICOULEAU C., Etude des problèmes d’optimisation dans les systèmes distribués, PhD Thesis, University of Paris 6, France, 1992.
- [RAV 87] RAVINDRAN A., PHILIPS D.T. and SOLBERG J.J., *Operations Research – Principles and Practice*, John Wiley, New York, 1987.
- [RAY 87] RAYWARD-SMITH V.J., “UET scheduling with interprocessor communication delays”, *Discrete Applied Mathematics*, vol. 18, p. 55–71, 1987.
- [SAN 05] SANLAVILLE E., “Sensitivity bounds for machine scheduling with uncertain communication delays”, *Journal of Scheduling*, vol. 8, p. 461–473, 2005.
- [SAN 07] SANLAVILLE E., “Sensitivity and robustness analyses in scheduling: the two machine with communication case”, p. 253–268, in ROY B., ALOULOU M.A. and KALAI R. (Eds.): *Robustness in OR-DA*, Paris, Annales du Lamsade, 2007.
- [SOT 98] SOTSKOV Y.N., WAGELMANS A.P.M. and WERNER F., “On the calculation of the stability radius of an optimal or an approximate schedule”, *Annals of Operational Research*, vol. 83, p. 213–225, 1998.
- [SOY 73] SOYSTER A.I., “Convex programming with set-inclusive constraints and applications to inexact linear programming”, *Operations Research*, vol. 21, p. 1154–1157, 1973.
- [VAR 96] VARVARIGOU T.A., ROYCHOWDHURY V.P., KAILATH P. and LAWLER E., “Scheduling in and out forests in the presence of communication delays”, *IEEE Trans. on Par. and Distr. Systems*, vol. 7, no. 10, 1996.
- [VEL 93] VELDHORST M., A linear time algorithm for scheduling trees with communication delays optimally on two machines, Report no. RUU-CS-93-04, Dept. of Computer Sciences, Utrecht, The Netherlands, 1993.
- [WAL 00] WALLACE S.W., “Decision making under uncertainty: is sensitivity analysis of any use?”, *Operations Research*, vol. 48, no. 1, p. 20–25, 2000.

Chapter 5

Service Level in Scheduling

5.1. Introduction

This chapter is concerned with stochastic scheduling problems in which various parameters are not known with certainty. However, we suppose that information is available on the probability laws followed by these parameters. The goal is to *explicitly* take them into account in scheduling problems, instead of implicitly as most of the deterministic proactive and/or reactive approaches do in this book. Even though information on probabilities of the various stochastic parameters may be difficult to obtain in practice, information systems, increasingly present in companies, allow starts and ends of tasks to be traced and stored. Analyzing these data, when they are available for several years, helps to determine the probability laws of some events and to estimate the parameters of these laws.

In this chapter, we study the case where processing times are stochastic, which is the most frequently studied in the literature. This is a rather general case since it includes possible failures on machines or varying processing speeds between operators that can be assigned to an operation but that are often not explicitly considered in scheduling problems.

A first important comment is that the literature on stochastic scheduling is relatively limited compared to the research on deterministic scheduling. Most of the

Chapter written by Stéphane DAUZÈRE-PÉRÈS, Philippe CASTAGLIOLA and Chams LAHLOU.

research on stochastic scheduling deals with the optimization of the mathematical expectation of criteria considered in deterministic problems. For project scheduling or flow-shop (the most studied stochastic scheduling problem), papers by Brucker *et al.* [BRU 99] and Gourgand *et al.* [GOU 03] show that the makespan is the most frequent criterion. For single or parallel machines, several criteria have been investigated: number of late jobs [JAN 02b], weighted number of late jobs [JAN 02a], sum of weighted completion times [LI 98, SKU 02, UET 03], earliness-tardiness [JIA 01, CAI 99] for instance. We refer the reader to the book by Pinedo [PIN 02] which contains a relatively recent survey on the optimization of the mathematical expectation for numerous criteria.

Optimizing the mathematical expectation can be seen as the transposition of the deterministic problem to the stochastic case. On the other hand, the idea of *service level*, introduced and studied in this chapter, and which corresponds to the R_5 measure of Chapter 1, has no equivalent in a deterministic problem.

In order to obtain analytical results, it is usual to assume that the operation processing times follow particular distribution functions, which make the mathematical analysis easier. In many instances, as noted in [PIN 02], the exponential distribution is used, which implies that the probability of an operation completing does not depend on the time the operation has been processed so far. Obviously, this is not a realistic assumption, particularly for production systems. The results given in sections 5.3 and 5.4 are based on more general and realistic distributions, such as gamma, lognormal, Weibull and beta distributions, which are characterized by a mean, a standard deviation, a minimum value and possibly a maximum value.

We study the notion of service level, to which very few papers have been dedicated, except those of Golenko-Ginzburg *et al.* [GOL 95] and Daniels and Carrillo [DAN 97].

Golenko-Ginzburg *et al.* [GOL 95] study the case of a job-shop problem where each job has a due date, a probability to be finished on time, and a weight representing its importance. Operations have random processing times, but their mathematical expectations and variances are known. They address two problems: in the first case, the goal is to maximize the sum of probabilities (for the jobs) to be on time; in the second case, the objective is to find a schedule minimizing the makespan, with the additional constraint that the probability of a job being on time is at least equal to a given value, i.e. a minimum service level is respected. The authors present several list scheduling algorithms which are tested on 5 machines and 10 jobs.

Daniels and Carrillo [DAN 97] are interested in a single machine problem, when jobs have random processing times with mathematical expectations and variances. They address the problem of finding a sequence for jobs that maximizes the probability a criterion associated with the sequence remains below a given threshold, i.e. a given service level. They propose a branch-and-bound algorithm and a heuristic for the sum of completion times ($\sum C_j$). Different problems of 10, 15 and 20 jobs are tested.

We will see in section 5.2 that the notion of service level, which answers questions like “what is the probability for this schedule to be completed before this time?”, may be interesting at several levels, such as the criterion to optimize or the constraints to satisfy.

Scheduling problems which take a service level into account are complex. It is generally difficult to compute a service level, even when the schedule is given, i.e. the sequencing and allocation of jobs are known in advance. For instance, in the case of a stochastic scheduling project (i.e. jobs must satisfy precedence constraints and processing times follow independent and discrete distributions) Hagstrom [HAG 88] proved that computing the probability to complete the project before a given date is an NP-complete problem. The mathematical expectation of a criterion takes into account only the mean of the processing times, whereas the service level integrates the range of processing times. Experimental results of section 5.3.3 emphasize that the range of processing times has a strong impact on the service level, and hence on the quality of a schedule.

This chapter aims to study, define and spur further research in the calculation of the service level associated with a given criterion and for a given schedule, but also the determination of a schedule that optimizes a given service level. Computing the service level of a given schedule can be seen as a means to perform sensitivity analysis, whereas optimizing a service level corresponds to optimizing robustness. The approaches that have been developed (and that mostly remain to be developed) to optimize service levels are primarily proactive approaches, since they search for the most robust schedule before it actually starts.

5.2. Motivations

To explain the notion of service level and its practical value in scheduling, it is relevant to compare inventory theory with stochastic demand. In this case, the main problem, if the order quantity is fixed, is often to determine the safety stock, i.e. the order point. This last parameter corresponds to the stock level below which it

is necessary to start a new order. The larger the order point, the larger the safety stock becomes. The order point is generally determined in two different ways (see for instance Silver *et al.* [SIL 98]):

- by minimizing the mean of a global cost (usually holding cost, order cost and backorder cost);
- or by satisfying a given service level.

The service levels are, for example, the probability that a backorder occurs between two replenishments or the percentage of demands satisfied without backorder. From a practical point of view, using a service level is often very interesting for several reasons. In particular, it may be very difficult to estimate backorder costs since they must take into account the potential loss of customers that are not satisfied (in addition to the costs related to sending products through faster transport means). It is then much easier to explain that an order point has been determined, to ensure that 99% of the demands are satisfied without backorders, than to minimize a global cost.

The service level in scheduling corresponds to the probability that a criterion is smaller (or larger) than or equal to a given value. If the makespan is considered, it is the probability that the makespan is smaller than or equal to a maximum time given to execute all the operations. It is thus possible to consider the service level of any criterion in deterministic scheduling. The criteria chosen to illustrate the results in this chapter are the makespan C_{\max} and the sum of the completion times $\sum C_j$. We believe that optimizing a service level, i.e. finding a schedule that maximizes the probability that a given quality is attained, allows the robustness to be optimized. As in inventory theory, this optimization criterion is relevant and easy to grasp. For example, it is easier to understand that the proposed schedule maximizes the chance that all the operations will be completed before the end of the day than that it minimizes the mean of the completion time of all the operations. The numerical results of section 5.3.3 show two things rather clearly:

- two schedules that seem equivalent, because they have the same mean, may have very different service levels;
- the schedule that minimizes the mean is not always the one that optimizes the service level.

This can be explained by the fact that, in some schedules, the variability of the operations has little or no impact on the service level: in the case of project scheduling for instance, if the operations with high variability do not belong to the critical path, the service level will be larger than in the opposite case.

Even if optimizing is not the goal, providing the service level for a schedule computed with deterministic parameters is very relevant from a practical point of view. For example, if a set of production orders must be manufactured in a given period (e.g. a day or a week), it may be very valuable to know that, using the schedule determined with any method, the probability of completing all the production orders on time is 95%.

Another interesting problem would be to optimize the mean of a criterion, but with the constraint that a given service level must be satisfied, as in the second problem considered by Golenko-Ginzburg *et al.* [GOL 95]. It would also be interesting to solve a multicriteria problem in which both the mean and the service level would be considered as objectives. The decision-maker could then define the trade-off between the “pure” performance of the schedule and its robustness. This problem is quite close to the one considered in Chapter 9, in which the robustness (service level) is replaced with flexibility.

5.3. Optimization of the service level: application to the flow-shop problem

We are interested in the flow-shop problem with m machines and n jobs (see Chapter 1 for a definition). Only permutation schedules are considered, i.e. when jobs are processed on each machine according to the same order. Given a permutation π , it is possible, using simulation, to estimate the mathematical expectation of the makespan $C_{\max}(\pi)$ and of the sum of job completion times $\sum C_j(\pi)$. In order to find a permutation minimizing the mathematical expectation, or maximizing the service level, these computations are done for each possible permutation. However, this approach, which enumerates all cases, is only possible for small problems such as the ones we will study: two machines and two jobs (two permutations), and three machines and five jobs (120 permutations).

5.3.1. Criteria computation

Let $X_{i,j}$ be the random variable corresponding to the processing time of job j on machine i . In our case, for a particular permutation π , the computation of the criteria is equivalent to the computation of the longest paths in a precedence graph where the values associated with the nodes are the $X_{i,j}$ s. As depicted in Figure 5.1, the value of the longest path from $X_{1,1}$ to $X_{m,n}$ corresponds to the value of $C_{\max}(\pi)$. For the same reason, the date corresponding to the end of job j is equal to the length of the longest path from $X_{1,1}$ to $X_{m,j}$.

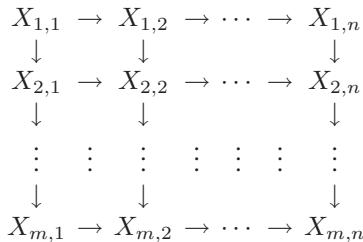


Figure 5.1. Precedence graph corresponding to permutation $(1, 2, \dots, n)$

The makespan ($C_{\max}(\pi)$) can be computed in the following way:

- $C_{1,1} = X_{1,1}$,
- For $j = 2 \dots n$, $C_{1,j} = C_{1,j-1} + X_{1,j}$,
- For $i = 2 \dots m$
 - $C_{i,1} = C_{i-1,1} + X_{i,1}$,
 - for $j = 2 \dots n$, $C_{i,j} = \max(C_{i,j-1}, C_{i-1,j}) + X_{i,j}$,
- $C_{\max}(\pi) = C_{m,n}$.

In the case of the sum of job completion times ($\sum C_j(\pi)$), the computation can be done in a similar way:

- $S = 0$ /* S represents the objective function */
- $C_{1,1} = X_{1,1}$,
- For $i = 2 \dots m$, $C_{i,1} = C_{i-1,1} + X_{i,1}$
- $S = S + C_{m,1}$,
- For $j = 2 \dots n$
 - $C_{1,j} = C_{1,j-1} + X_{1,j}$,
 - for $i = 2 \dots m$, $C_{i,j} = \max(C_{i,j-1}, C_{i-1,j}) + X_{i,j}$,
 - $S = S + C_{m,j}$,
- $\sum C_j(\pi) = S$.

5.3.2. Processing time generation

We will assume that for each random variable $X_{i,j}$ we know the mean μ , the standard deviation $\sigma > 0$, the minimal value c and the definition interval length $d > 0$ such that $X_{i,j}$ is defined on $[c, c + d]$. In this chapter, we will focus on four different types of probability distributions: the gamma distribution, the lognormal distribution,

the Weibull distribution and the beta distribution. Each of these distributions depends on two parameters: a (shape) and b (scale). These probability distribution were chosen because they have quite different shapes and are easy to simulate. For each of the four distributions, we will define the PDF (*probability density function*) and we will give the relations between the distribution parameters (a, b) and the input parameters (μ, σ, c, d) :

- The PDF of the gamma (a, b, c) distribution is defined on $[c, +\infty)$, i.e. $d = +\infty$, and is equal to

$$f_\gamma(x|a, b, c) = \frac{(x - c)^{a-1} \exp\left(\frac{c-x}{b}\right)}{b^a \Gamma(a)}$$

where $\Gamma(a)$ is the gamma function. The values of a and b are equal to

$$\begin{aligned} a &= \left(\frac{\mu - c}{\sigma}\right)^2 \\ b &= \frac{\mu - c}{a} \end{aligned}$$

- The PDF of the lognormal (a, b, c) distribution is defined on $[c, +\infty)$, i.e. $d = +\infty$, and is equal to

$$f_L(x | a, b, c) = \frac{bf_N(a + b \ln(x - c))}{x - c}$$

where $f_N(t) = (2\pi)^{-1/2} \exp(-t^2/2)$ is the normal $(0, 1)$ PDF. In order to compute a and b , we first have to evaluate

$$\begin{aligned} u &= \left(1 + \left(\frac{\sigma}{\mu - c}\right)^2\right)^{1/2} \\ v &= \frac{\mu - c}{u} \end{aligned}$$

and then

$$b = (2 \ln u)^{-1/2}$$

$$a = -b \ln v$$

– The PDF of the Weibull (a, b, c) distribution is defined on $[c, +\infty)$, i.e. $d = +\infty$, and is equal to

$$f_W(x | a, b, c) = \frac{a}{b} \left(\frac{x - c}{b} \right)^{a-1} \exp \left(- \left(\frac{x - c}{b} \right)^a \right)$$

In order to find the value of a , the following non-linear equation has to be solved

$$\frac{\Gamma(1 + 2/a)}{\Gamma^2(1 + 1/a)} = 1 + \left(\frac{\sigma}{\mu - c} \right)^2$$

and then

$$b = \frac{\mu - c}{\Gamma(1 + 1/a)}$$

– The PDF of the beta (a, b, c, d) distribution is defined on $[c, c + d]$ and is equal to

$$f_\beta(x | a, b, c, d) = \frac{(x - c)^{a-1}(d - x)^{b-1}}{B(a, b)(d - c)^{a+b-1}}$$

where $B(a, b)$ is the beta function. In order to compute a and b , we must first compute $\mu' = (\mu - c)/d$ and $\sigma' = \sigma/d$ and then

$$a = \frac{\mu'^2(1 - \mu')}{\sigma'^2} - \mu'$$

$$b = \frac{\mu'(1 - \mu')}{\sigma'^2} - 1 - a$$

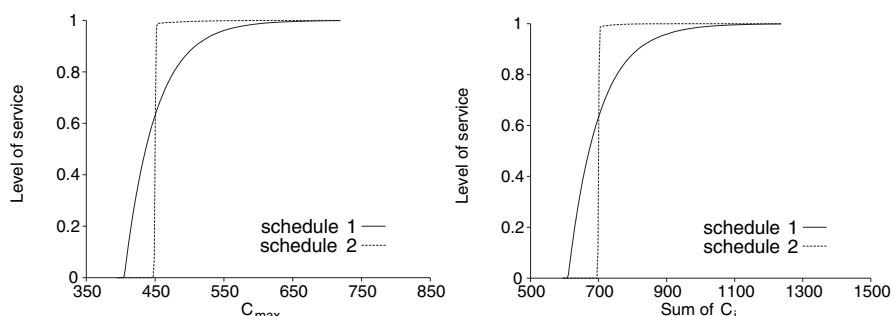
5.3.3. Experimental results

The simple case of a flow-shop problem with two machines and two jobs is first considered. We assume that the processing times of operations follow the same distribution, which means that the problem parameters c, μ, σ (and d for the beta distribution case) are the same irrespective of the chosen distributions (see Table 5.1). Note that only the processing time of the first operation of the first job (i.e. $X_{1,1}$) can vary significantly.

	c	μ	σ	d
$X_{1,1}$	5	50	45	95
$X_{2,1}$	195	200	0.5	205
$X_{1,2}$	45	50	0.5	55
$X_{2,2}$	195	200	0.5	205

Table 5.1. Values of the parameters for 2×2 flow-shops

For these examples, the two schedules $\pi_1 = (1, 2)$ and $\pi_2 = (2, 1)$ have almost the same mathematical expectation of the chosen criterion ($E(C_{\max}(\pi_1)) = 450$ and $E(C_{\max}(\pi_2)) = 451$; $E(\sum C_j(\pi_1)) = 699$ and $E(\sum C_j(\pi_2)) = 701$). However, as Figure 5.2 shows, in the case of the gamma distribution, they do not have the same service level at all: the gap can reach 35%. For instance, the probability of having a makespan less than 452 is 65% for the schedule minimizing the mathematical expectation of the makespan, whereas it is 98% for the other one. Therefore, a threshold exists below which one of the schedules has the best service level, and above which the other schedule has, on the contrary, a better service level. This can be explained when the criterion is expressed in terms of random variables. The makespans of the two schedules are $C_{\max}((1, 2)) = X_{1,1} + \max\{X_{1,2}, X_{2,1}\} + X_{2,2}$ and $C_{\max}((2, 1)) = X_{2,1} + \max\{X_{2,2}, X_{1,1}\} + X_{1,2}$. Hence, $C_{\max}((1, 2))$ is very dependent on $X_{1,1}$, which varies considerably, whereas $C_{\max}((2, 1))$ is very stable because the value of $X_{1,1}$ is dominated by that of $X_{2,2}$.

**Figure 5.2.** 2×2 flow-shop (gamma distribution): same mathematical expectation of the criteria

The same curves are observed when the distributions are lognormal and Weibull. If the distribution is bounded, such as the beta distribution, the gap between the service levels is larger (it can reach 50%; see Figure 5.3).

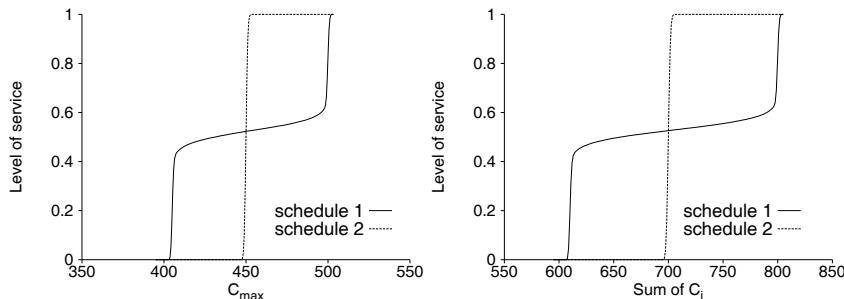


Figure 5.3. 2×2 flow-shop (beta distribution):
same mathematical expectation of the criteria

The same phenomenon can be observed in case of a flow-shop with three machines and five jobs and the operations follow the same distribution. In Figure 5.4, in the case of the lognormal distribution, we can see the service level of a schedule minimizing the mathematical expectation of the criterion, and an optimal service level, obtained by taking the maximum service level among the 120 schedules, for each value of the criterion.

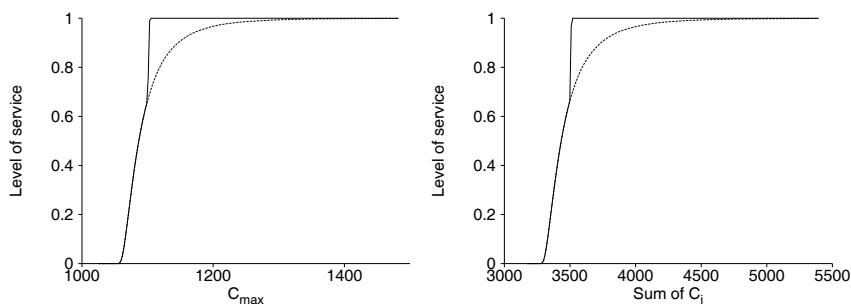


Figure 5.4. 3×5 flow-shop (lognormal distribution):
same mathematical expectation of the criteria

Again, all schedules have almost the same mathematical expectation of the chosen criterion (from $\min_{\pi}(E(C_{\max}(\pi))) = 1099.5$ to $\max_{\pi}(E(C_{\max}(\pi))) = 1101.2$, and from $\min_{\pi}(E(\sum C_j(\pi))) = 3498.9$ to $\max_{\pi}(E(\sum C_j(\pi))) = 3505$) while they have service levels that can be very different (the gap can be 35%). Moreover, several

thresholds exist (15 for C_{\max} , and 20 for $\sum C_j$) which correspond to the different schedules with the best service level.

Finally, we compared the flow-shop problem with two machines and two jobs when the mathematical expectations are different: on the one hand $E(C_{\max}(\pi_1)) = 325$ and $E(C_{\max}(\pi_2)) = 357$, on the other hand $E(\sum C_j(\pi_1)) = 500$ and $E(\sum C_j(\pi_2)) = 554$. Figure 5.5 shows, in the case of the lognormal distribution, that the schedule minimizing the mathematical expectation has a 10% lower service level after a certain threshold.

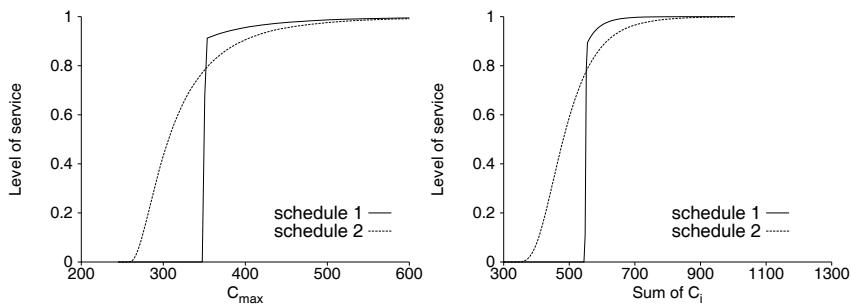


Figure 5.5. *2×2 flow-shop (lognormal distribution): different mathematical expectation of the criteria*

5.4. Computation of a schedule service level

Computing the service level of a given schedule (i.e. when the sequencing and the allocation of operations are known) can be seen as working on a precedence graph, that is, considering a project scheduling problem. When the criterion is the makespan, the problem is similar to computing the distribution function of the project duration. In this case, we can use numerous methods (see the surveys [ADL 87, CUB 92, BAC 93] for example). Basically, there are three main approaches:

- Simulation (mainly Monte Carlo) which gives an approximate value of the distribution function of the project duration.
- Analysis, based on critical paths or the graph structures (such as series-parallels).
- Computation of upper and lower bounds of the distribution function of the project duration.

Additionally, the reader is referred to the paper by Iida [IID 00], who introduces upper and lower bounds based on a path analysis in the graph, and the paper by Möhring [MÖH 01], who presents several bounds based on a graph transformation.

On the other hand, to our knowledge, there exist no works on the computation of the distribution function for criteria other than the makespan for the project scheduling problem. Even for deterministic problems, little research has been done, as the surveys of Brucker *et al.* [BRU 99] and of Klein [KLE 99] show.

5.4.1. Introduction

Let X_1, \dots, X_p be p independent continuous random variables. Let $f_{X_i}(x)$ and $F_{X_i}(x)$ be respectively the PDF (probability density function) and the CDF (cumulative distribution function) of the random variable X_i . Let $z(x_1, \dots, x_p)$ be a function from $\mathbb{R}^p \rightarrow \mathbb{R}$. In our case, the function $z(x_1, \dots, x_p)$ corresponds to one of the considered criterion, i.e. C_{\max} and $\sum C_j$ as defined in section 5.3. By definition, the CDF $F_Z(z)$ of the random variable $Z = z(X_1, \dots, X_p)$ is equal to

$$F_Z(z) = P\{z(X_1, \dots, X_p) \leq z\} = \int_{z(x_1, \dots, x_p) \leq z} f_{X_1}(x_1) \cdots f_{X_p}(x_p) dx_1 \cdots dx_p.$$

The computation of this multivariate integral is a complex problem and, with the exception of some specific cases for which the function $z(x_1, \dots, x_p)$ and/or the PDFs $f_{X_1}(x), \dots, f_{X_p}(x)$ have a particular structure, there is no general closed-form solution. Due to the potentially large number p of variables, the numerical evaluation (quadrature) of this multivariate integral is also impossible. In order to obtain an approximation $\hat{F}_Z(z)$ of $F_Z(z)$, a classical approach consists of using a Monte Carlo simulation, i.e. randomly generate N replicates $x_i^{(1)}, \dots, x_i^{(N)}$ for each random variable X_i and then compute the following N values

$$\begin{aligned} z^{(1)} &= z(x_1^{(1)}, \dots, x_p^{(1)}) \\ &\vdots \vdots \vdots \\ z^{(N)} &= z(x_1^{(N)}, \dots, x_p^{(N)}) \end{aligned}$$

An approximation $\hat{F}_Z(z)$ of $F_Z(z)$ is then equal to

$$\hat{F}_Z(z) = \frac{\#(z^{(k)} \leq z)}{N}$$

where $\#(z^{(k)} \leq z)$ is the number of $z^{(i)}$'s among $z^{(1)}, \dots, z^{(N)}$ having a value smaller than or equal to z . The Monte Carlo method is a computer intensive method that gives good results if the number N of replicates is sufficiently large. Moreover, the

larger the number p of variables, the larger the number N of replicates in order to keep a good accuracy when computing $\hat{F}_Z(z)$. If the number p of variables becomes too large, the Monte Carlo method (or variants of this method) tends to be too time consuming without being accurate enough. In order to overcome this major drawback, some specific methods were developed. One of them is called FORM (First Order Reliability Method).

5.4.2. FORM (First Order Reliability Method)

This method is derived from the structural reliability research field. For instance, in this field, the rigidity z of a metallic structure depends on a large number of structural variables x_1, \dots, x_p . As long as the rigidity of the structure is above a predefined threshold, it has a normal behavior while, if the rigidity falls below the predefined threshold, the structure is likely to be deformed or even broken. If the CDF associated with each structural variable (assumed to be random variables) is known, the FORM method helps to approximate the CDF of the rigidity and then to evaluate the probability that the rigidity remains above a predefined threshold. This method originates with the works of Freudenthal [FRE 56] but its modern developments are mainly due to Hasofer and Lind [HAS 74], Rackwitz and Fiessler [RAC 78] and Breitung [BRE 84]. An important reference on this topic is the work of Melchers [MEL 99]. The base idea of the FORM method is to transform (see Figure 5.6) the p random variables X_1, \dots, X_p into p new random variables U_1, \dots, U_p using the Rosenblatt's transformation [ROS 52]

$$U_k = F_N^{-1}(F_{X_k}(X_k)) \quad (5.1)$$

where $F_N^{-1}(t)$ is the inverse cdf of the normal $(0, 1)$ distribution. From equation (5.1), we have $X_k = F_{X_k}^{-1}(F_N(U_k))$ and consequently

$$\frac{d}{du_k} F_{X_k}^{-1}(F_N(u_k)) = \frac{f_N(u_k)}{f_{X_k}(F_{X_k}^{-1}(F_N(u_k)))}$$

We finally obtain

$$F_Z(z) = \int_{h(u_1, \dots, u_p) \leq z} f_N(u_1) \cdots f_N(u_p) du_1 \cdots du_p$$

with

$$h(u_1, \dots, u_p) = z(F_{X_1}^{-1}(F_N(u_1)), \dots, F_{X_n}^{-1}(F_N(u_p)))$$

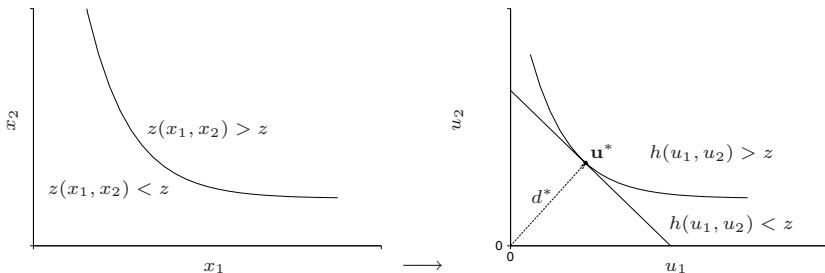


Figure 5.6. Rosenblatt's transformation in the case of $p = 2$ variables

Thus, the Rosenblatt's transformation creates a new space where U_1, \dots, U_p are independent normal $(0, 1)$ random variables. The key to the FORM method consists of finding the point $\mathbf{u}^* = (u_1^*, \dots, u_p^*)^T$ of \mathbb{R}^p minimizing the Euclidean distance $|\mathbf{u}| = (\mathbf{u}^T \mathbf{u})^{1/2}$ to the origin under the constraint $h(u_1, \dots, u_p) = z$. The point \mathbf{u}^* is called the MPP or most probable point. Using the properties of the multinormal distribution, if $d^* = (\mathbf{u}^{*T} \mathbf{u}^*)^{1/2}$ is the Euclidean distance between the point \mathbf{u}^* and the origin, then an approximation $\hat{F}_Z(z)$ of $F_Z(z)$ is equal to

$$\hat{F}_Z(z) = \begin{cases} F_N(d^*) & \text{if } u_1 \geq 0, \dots, u_p \geq 0 \\ F_N(-d^*) & \text{otherwise.} \end{cases} \quad (5.2)$$

This approximation is equivalent to the exact computation of the probability that the random variables U_1, \dots, U_p belong to the region “below” the hyperplane passing through the point \mathbf{u}^* and tangent to the hypersurface of equation $h(u_1, \dots, u_p) = z$. The approximation corresponding to equation (5.2) will be better if the hypersurface of equation $h(u_1, \dots, u_p) = z$ is close to a hyperplane. A possible but more complex extension of the FORM method is called the SORM (Second Order Reliability Method) initially developed by Breitung [BRE 84] and improved by Tvedt [TVE 90]. This method is identical to the FORM method with regard to Rosenblatt's transformation. The difference is the fact that the SORM method takes into account the curvature of the hypersurface of equation $h(u_1, \dots, u_p) = z$ at point \mathbf{u}^* .

5.4.3. FORM vs Monte Carlo

The FORM and Monte Carlo methods are implemented in C and tested on a PC (Pentium III, 700 MHz)¹. The number N of replicates used during the Monte Carlo simulation is 100,000.

1. These experiments were conducted by Mohammed Brahimi during his Master's thesis [BRA 04].

To begin with, we consider a flow-shop with two machines and two jobs. As was the case in section 5.3.3, we assume that the processing times of operations follow the same distribution (i.e. the problem parameters c, μ, σ (and d for the beta distribution case) are the same: see Table 5.1).

The results for the lognormal distribution are reported in Table 5.2 and Figure 5.7. We notice that FORM does not converge for small values of C_{\max} and $\sum C_j$: $y < 405$ for the C_{\max} case, and $y < 610$ for the $\sum C_j$ case. On the contrary, FORM converges in few iterations if $405 \leq y \leq 765$ for the C_{\max} case, and if $610 \leq y \leq 1325$ for the $\sum C_j$ case.

C_{\max}			$\sum C_j$		
y	FORM	Monte Carlo	y	FORM	Monte Carlo
405	0.000009	0.000000	610	0.000003	0.000000
425	0.289692	0.282500	630	0.083358	0.085000
450	0.661854	0.671667	650	0.289376	0.282500
475	0.828373	0.832500	700	0.661743	0.670833
500	0.905636	0.914167	750	0.828325	0.833333
525	0.944625	0.952500	800	0.905612	0.914167
550	0.965777	0.970000	850	0.944613	0.952500
575	0.977946	0.982500	950	0.977942	0.982500
600	0.985290	0.985000	1050	0.989900	0.989167
765	0.998216	1.000000	1325	0.998168	1.000000

Table 5.2. 2×2 flow-shop (lognormal distribution): service level by Monte Carlo and FORM

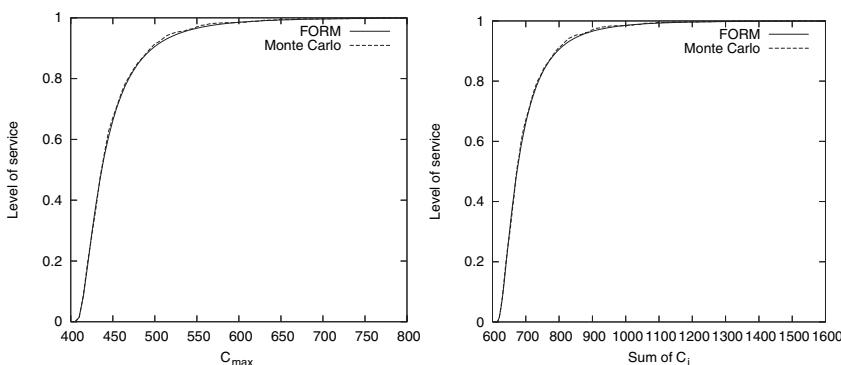


Figure 5.7. 2×2 flow-shop (lognormal distribution)

Table 5.3 and Figure 5.8 report similar results for the gamma distribution. As for the previous distribution, FORM does not converge for small values of C_{\max} and $\sum C_j$: if $y < 405$ for the C_{\max} case, if $y < 610$ for the $\sum C_j$ case. The method converges in few iterations otherwise. For both distributions, FORM and Monte Carlo compute almost the same service level (see Figures 5.7 and 5.8).

C_{\max}			$\sum C_j$		
y	FORM	Monte Carlo	y	FORM	Monte Carlo
405	0.009066	0.005000	610	0.007167	0.004167
425	0.359343	0.365000	630	0.199855	0.195833
450	0.632378	0.622500	650	0.359206	0.361667
475	0.789069	0.786667	700	0.632315	0.623333
500	0.878976	0.880000	750	0.789036	0.788333
525	0.930561	0.926667	800	0.878957	0.881667
550	0.960159	0.959167	850	0.944613	0.952500
575	0.977141	0.975833	950	0.977138	0.975833
600	0.986885	0.987500	1050	0.992474	0.995000
700	0.998579	1.000000	1200	0.998578	1.000000

Table 5.3. 2×2 flow-shop (gamma distribution): service level by Monte Carlo and FORM

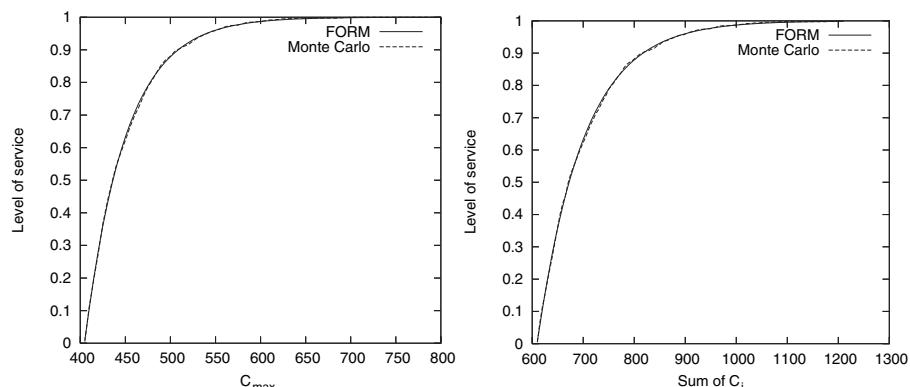


Figure 5.8. 2×2 flow-shop (gamma distribution)

In the case of a beta distribution, our tests show that FORM does not converge for small and large values of the criteria: $y < 405$ and $y > 499$ for the C_{\max} case; $y < 610$ and $y > 795$ for the $\sum C_j$ case. There is a small gap between the service

C_{\max}			$\sum C_j$		
y	FORM	Monte Carlo	y	FORM	Monte Carlo
400	-	0.000000	605	-	0.000000
405	0.363757	0.200833	610	0.359172	0.194167
410	0.454192	0.465000	620	0.454150	0.465833
420	0.483777	0.493333	630	0.472227	0.480000
430	0.500033	0.509167	650	0.492610	0.502500
440	0.512589	0.524167	670	0.506571	0.520000
460	0.534666	0.552500	700	0.523743	0.538333
480	0.560215	0.570000	750	0.552831	0.567500
499	0.635754	0.675000	795	0.614964	0.623333
503	-	1.000000	805	-	1.000000

Table 5.4. 2×2 flow-shop (beta distribution): service level by Monte Carlo and FORM

level computed by FORM and the one given by Monte Carlo simulation, for both criteria (see Figure 5.9).

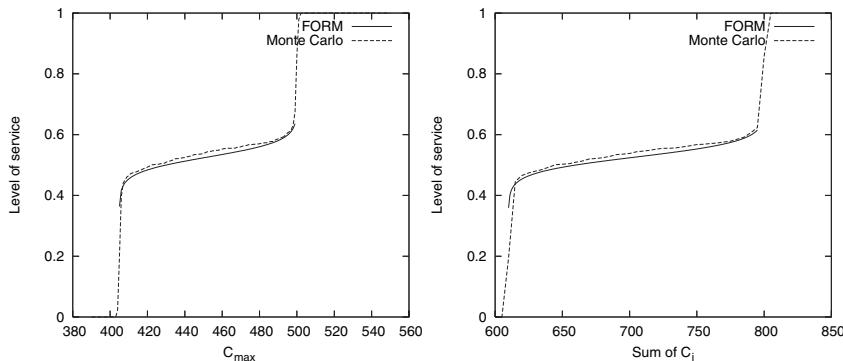


Figure 5.9. 2×2 flow-shop (beta distribution)

Next, we consider more machines and jobs to check if the methods have the same behavior, i.e. good results with the lognormal and gamma distributions and a small gap with the beta distribution.

A flow-shop with three machines and five jobs is considered. Here again the processing times of operations follow the same distribution (see Table 5.5).

	c	μ	σ	d
$X_{1,1}$	5	50	45	95
$X_{2,1}$	45	50	0.5	55
$X_{3,1}$	195	200	0.5	205
$X_{1,2}$	45	50	0.5	55
$X_{2,2}$	45	50	0.5	55
$X_{3,2}$	195	200	0.5	205
$X_{1,3}$	45	50	0.5	55
$X_{2,3}$	45	50	0.5	55
$X_{3,3}$	195	200	0.5	205
$X_{1,4}$	45	50	0.5	55
$X_{2,4}$	45	50	0.5	55
$X_{3,4}$	195	200	0.5	205
$X_{1,5}$	45	50	0.5	55
$X_{2,5}$	45	50	0.5	55
$X_{3,5}$	195	200	0.5	205

Table 5.5. Values of the parameters for 3×5 flow-shops

In the case of lognormal or gamma distributions, the results show that for small values of C_{\max} and $\sum C_j$ the FORM method does not converge. The experiments also show that Monte Carlo and FORM give the same service level (see Figures 5.10 and 5.11). Hence, the size of the problem does not seem to have an impact on the results.

C_{\max}			$\sum C_j$		
y	FORM	Monte Carlo	y	FORM	Monte Carlo
1055	0.001424	0.000000	3270	0.000054	0.000000
1085	0.474301	0.473333	3330	0.106023	0.111667
1100	0.662498	0.671667	3340	0.145601	0.158333
1130	0.848818	0.849167	3380	0.311523	0.309167
1160	0.924369	0.915833	3420	0.457344	0.461667
1190	0.958782	0.955833	3480	0.620596	0.629167
1240	0.982784	0.983333	3550	0.745080	0.743333
1265	0.985951	0.987500	4150	0.979725	0.977500
1405	0.990733	0.999167	4530	0.988914	0.992500
1500	0.992892	1.000000	4700	0.990601	0.997500

Table 5.6. 3×5 flow-shop (lognormal distribution): service level by Monte Carlo and FORM

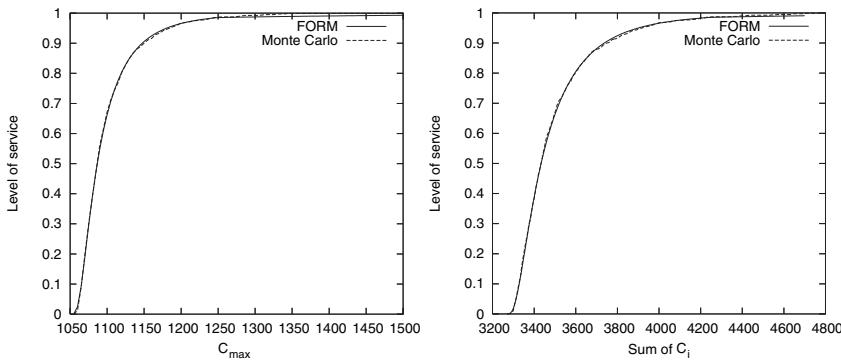


Figure 5.10. 3×5 flow-shop (lognormal distribution)

C_{\max}			$\sum C_j$		
y	FORM	Monte Carlo	y	FORM	Monte Carlo
1055	0.030696	0.007500	3270	-	0.000833
1080	0.427762	0.431667	3290	0.071267	0.060833
1100	0.632742	0.640833	3305	0.128589	0.127500
1130	0.811357	0.821667	3375	0.360009	0.373333
1165	0.913314	0.920000	3435	0.509619	0.525000
1200	0.960170	0.955000	3470	0.580215	0.585000
1230	0.979550	0.975833	3635	0.798328	0.807500
1355	0.991771	0.998333	3790	0.898726	0.905000
1455	0.994724	1.000000	4380	0.989178	0.993333
1500	0.995680	1.000000	4530	0.991334	0.996667

Table 5.7. 3×5 flow-shop (gamma distribution):
service level by Monte Carlo and FORM

For the beta distribution, FORM does not converge for small values of the criterion. Moreover there is a significant gap between the service levels of the two methods (see Figure 5.12).

Our experiments show that:

- if the processing times have a lognormal or gamma distribution, (i) FORM is a very good method to approximate the service level and (ii) the size of the problem does not seem to have an impact on the quality of the approximation;
- if the processing times have a beta distribution, (i) the approximation by FORM is worst for the 2×2 flow-shop, and (ii) can be bad when the size of the problem increases.

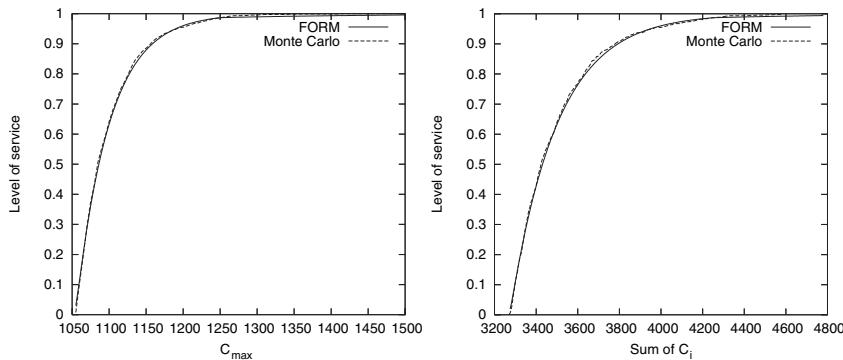


Figure 5.11. 3×5 flow-shop (gamma distribution)

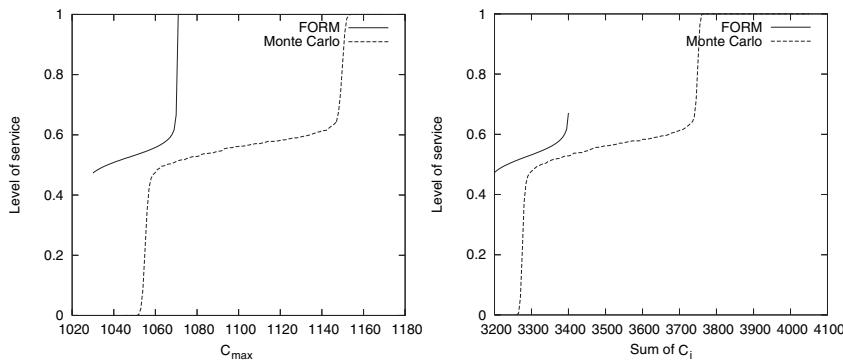


Figure 5.12. 3×5 flow-shop (beta distribution)

5.5. Conclusions

In this chapter, we introduced the notion of service level in stochastic scheduling. We have shown the relevance of this notion, something that has rarely been done in the scientific literature. Through various examples, it clearly appears that only optimizing the mean of a performance may lead to very poor service levels, even though a small degradation of the mean performance can induce a much improved service level.

We believe that two types of basic research considering service levels are particularly interesting:

- 1) the determination of a service level for a given schedule; and
- 2) the determination of a schedule that optimizes a service level.

Some research has been conducted on the first type of problems when the service level is associated with the makespan. However, there is very little research on service levels based on other criteria. The most interesting problems will probably concern the optimization of an average performance with the constraint of a minimum service level to satisfy. In the examples of this chapter and for the purpose of illustration, the complete distribution of the service level has been computed. However, it is important to note that it is *a priori* not necessary to know this complete distribution if the service level must be determined for one or several fixed limits, which is often the case in practice.

Optimizing service levels is much more difficult, and will probably be based on results obtained for the problems discussed above. It should be noted that there is nearly no research on the optimization of a service level. Another perspective is multicriteria optimization that simultaneously considers the mean and the service level.

The resolution approaches that should be developed are by definition proactive. Another research perspective would be to see how these approaches could be either extended to make them reactive or coupled with reactive approaches to be defined.

We believe that stochastic scheduling with service levels is a research field which is rich and promising, and of great practical interest. However, difficulties are numerous, since it involves combining complex combinatorial optimization problems with stochastic theory.

5.6. Bibliography

- [ADL 87] ADLAKHA V. and KULKARNI V., A classified bibliography of research on stochastic PERT networks: 1966-1987, Technical report, Dept. of Op. Res., University of North Carolina, Chapel Hill, 1987.
- [BAC 93] BACCELLI F., LIU Z. and JEAN-MARIE A., A survey on solution methods for task graph models, Report, Second QMIPS Workshop, Götz, Herzog, Rettelbach (Ed.), Arbeitsberichte der IMMD, 26, 14, Erlangen, March 1993, http://www-sop.inria.fr/mistral/info/QMIPS/Deliverables/D1/chapter3/INR-graph_survey.ps.gz.
- [BRA 04] BRAHIMI M., Approximation du niveau de service d'un ordonnancement stochastique, Master's Thesis, Ecole des Mines de Nantes/University of Nantes, September 2004.
- [BRE 84] BREITUNG K., "Asymptotic approximations for multinormal integrals", *ASCE Journal of Engineering Mechanics*, vol. 110, p. 357-366, 1984.

- [BRU 99] BRUCKER P., DREXL A., MÖHRING 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.
- [CAI 99] CAI X. and ZHOU S., “Stochastic scheduling on parallel machines subject to random breakdowns to minimize expected costs for earliness and tardy jobs”, *Operations Research*, vol. 47, no. 3, p. 422–437, 1999.
- [CUB 92] CUBAUD P., Modèles et outils pour la prédition des performances des systèmes informatiques parallèles, PhD Thesis, Université de Paris V, March 1992.
- [DAN 97] DANIELS R.L. and CARRILLO J.E., “ β -Robust scheduling for single-machines systems with uncertain processing times”, *IEE Transactions*, vol. 29, p. 977–985, 1997.
- [FRE 56] FREUDENTHAL A.N., “Safety and the probability of structural failure”, *Trans. ASCE*, vol. 121, p. 1337–1397, 1956.
- [GOL 95] GOLENKO-GINZBURG D., KESLER S. and LANDSMAN Z., “Industrial job-shop scheduling with random operations and different priorities”, *International Journal of Production Economics*, vol. 40, p. 185–195, 1995.
- [GOU 03] GOURGAND M., GRANGEON N. and NORRE S., “A contribution to the stochastic flow shop scheduling problem”, *European Journal of Operational Research*, vol. 151, no. 2, p. 415–433, 2003.
- [HAG 88] HAGSTROM J.N., “Computational complexity of PERT problems”, *Networks*, vol. 18, p. 139–147, 1988.
- [HAS 74] HASOFEK A.M. and LIND N.C., “An exact and invariant first order reliability format”, *ASCE Journal of Engineering Mechanics*, vol. 110, p. 111–121, 1974.
- [IID 00] IIDA T., “Computing bounds on project duration distributions for stochastic PERT networks”, *Naval Research Logistics*, vol. 47, p. 559–580, 2000.
- [JAN 02a] JANG W., “Dynamic scheduling of stochastic jobs on a single machine”, *European Journal of Operational Research*, vol. 138, p. 518–530, 2002.
- [JAN 02b] JANG W. and KLEIN C.M., “Minimizing the expected number of tardy jobs when processing times are normally distributed”, *Operations Research Letters*, vol. 30, p. 100–106, 2002.
- [JIA 01] JIA C., “Stochastic single machine scheduling with an exponentially distributed due date”, *Operations Research Letters*, vol. 28, p. 199–203, 2001.
- [KLE 99] KLEIN R., *Scheduling of Resource-Constrained Projects*, Kluwer Academic Publishers, Boston, USA, 1999.
- [LI 98] LI W. and GLAZEBROOK K.D., “On stochastic machine scheduling with general distributional assumptions”, *European Journal of Operational Research*, vol. 105, p. 525–536, 1998.

- [MEL 99] MELCHERS R.E., *Structural Reliability Analysis and Prediction*, John Wiley & Sons, NY, 2nd edition, 1999.
- [MÖH 01] MÖHRING R.H., “Scheduling under uncertainty: bounding the makespan distribution”, in ALT H. (Ed.), *Computational Discrete Mathematics: Advanced Lectures*, vol. 2122 of *Lecture Notes in Computer Science*, p. 79–97, Springer, 2001.
- [PIN 02] PINEDO M., *Scheduling: Theory, Algorithms, and Systems*, Prentice Hall, Upper Saddle River, NJ, 2nd edition, 2002.
- [RAC 78] RACKWITZ R. and FIESSLER B., “Structural reliability under combined load sequences”, *Computers & Structures*, vol. 9, p. 489–494, 1978.
- [ROS 52] ROSENBLATT M., “Remarks on a multivariate transformation”, *Annals of Mathematical Statistics*, vol. 23, no. 3, p. 470–472, 1952.
- [SIL 98] SILVER E., PYKE D. and PETERSON R., *Inventory Management and Production Planning and Scheduling*, John Wiley & Sons, NY, 3rd edition, 1998.
- [SKU 02] SKUTELLA M. and UETZ M., Stochastic Machine Scheduling with Precedence Constraints, Technical report no. 756-2002, Technische Universität Berlin, Germany, 2002.
- [TVE 90] TVEDT L., “Distribution of quadratic forms in normal space – application to structural reliability”, *ASCE Journal of Engineering Mechanics*, vol. 116, no. 6, p. 1183–1197, 1990.
- [UET 03] UETZ M., “When greediness fails: examples from stochastic scheduling”, *Operations Research Letters*, vol. 31, p. 413–419, 2003.

Chapter 6

Metaheuristics for Robust Planning and Scheduling

6.1. Introduction

As emphasized in Chapter 1, the stakes of robust scheduling are particularly high. However, many solution methods which have been developed, often within theoretical frameworks, are not easily applied in practice. Moreover, manufacturers, who are committed to working with researchers, are often incapable – partially or totally – of providing data that is reliable or that perfectly corresponds to the problem at hand. To try to compensate for this deficit we propose a method for robust optimization in this chapter.

Optimization problems are often \mathcal{NP} -hard, even when they are deterministic, i.e., when all data is assumed to be known with 100% certainty. There are, therefore, no efficient algorithms to optimally solve it. Metaheuristics have become a solution tool of preference, guaranteeing both reduced development time and a high-quality solution. An impressive number of articles using metaheuristics to solve scheduling problems can be found in the literature. Among the most valued techniques we find genetic algorithms and tabu search. [REE 97] presents a non-exhaustive list of genetic algorithm applications in different optimization areas. A more recent study presents a general panorama of metaheuristics [HAO 99].

Chapter written by Marc SEVAUX, Kenneth SÖRENSEN and Yann LE QUÉRÉ.

Although metaheuristics and particularly genetic algorithms are often successfully applied when dealing with scheduling problems, there are only a few applications of these techniques in an uncertain environment. In this chapter, we assert that metaheuristics can be easily adapted to the needs of a stochastic problem. Using metaheuristics for this type of optimization has a certain number of advantages which are presented throughout the chapter. The need for robust optimization in metaheuristics has been recognized in the well known book on robust optimization [KOU 97], when on page 354 the authors write that “*We believe that considerable more effort should be spent on the systematic development of [...] metaheuristic frameworks, which with minimal adjustment effort can be applied to a large class of robust optimisation problem [...]*”.

This chapter therefore presents a general framework for robust optimization (section 6.2) followed by two applications, in scheduling (section 6.3) and in planning (section 6.4). To put the chapter into the context of the book, we will only be using a proactive approach here. Providing such solutions has a very real interest when, on the one hand, there is considerable uncertainty in the problem data (section 6.3) or on the other, when social and historical constraints make the plan difficult to modify (section 6.4).

6.2. A general framework for metaheuristic robust optimization

The objective of this chapter is not to propose a single genetic algorithm to deal with robust scheduling and planning problems, but to present a general optimization framework in which metaheuristics can be systematically used to produce robust solutions. This section first presents the general principle of the method (section 6.2.1), and then a practical implementation of it in a genetic algorithm for robust optimization (section 6.2.2). The last two sections of this chapter present an application of the framework.

6.2.1. General considerations

For all metaheuristics, we must define both the encoding of the solutions to the problem and the evaluation function, which attaches to each solution a real-valued number indicating its quality. The method presented here transforms a metaheuristic for the deterministic optimization of a given problem, into one to find solutions in the stochastic case. As in [SÖR 01], this change is made by replacing the evaluation function with a *robust evaluation function*. If a distinction needs to be made between the two types of objective functions, the one for the deterministic case will be called *an ordinary evaluation function*. The rest of this section is devoted to the definitions

and explanations necessary for the preservation of the quality of the solution in the uncertain case.

Let s be a solution to our problem. The quality of solution s is computed by an evaluation function $z(s)$. When necessary, as defined in Chapter 1, we can specify which particular instance \mathcal{I} of problem \mathcal{P} we act upon, thus the evaluation function will be written $z_{\mathcal{I}}(s)$. To search for the robust solution, the evaluation function must be replaced by a *robust evaluation function* $f^r(s)$. To find robust solutions, this function adheres to the following two principles [SÖR 01, SÖR 03]:

Principle 1: Each solution is implemented on a set of characteristics or modified parameters $\zeta(\mathcal{I})$ of the initial instance \mathcal{I} . ζ is a *sampling function* which takes a random sample of the uncertain elements of \mathcal{I} . Let $\zeta_i(\mathcal{I})$ be the i th sample of parameters of instance \mathcal{I} . The implementation of a solution on a modified set of parameters is called a *derived solution*.

Principle 2: Several evaluations of a solution on a sample of \mathcal{I} are combined into a new evaluation function. An evaluation of a derived solution is called a *derived evaluation*. This new function, which uses the same principles as [KOU 97], is the *robust evaluation function* $f^r(s)$.

A possible form of a robust evaluation function is the average of m derived evaluations:

$$f^r(s) = \frac{1}{m} \sum_{i=1}^m z_{\zeta_i(\mathcal{I})}(s) \quad (6.1)$$

where m is the number of derived solutions to be evaluated. This is the same as criterion $R3$ defined in Chapter 1 and similar to the evaluation function used in Chapter 7 where the evaluation is carried out in an iterative manner and not averaged, as here.

A more conservative form of a robust evaluation function consists in examining the worst case of a solution among all the derived evaluations (worst-case analysis):

$$f^r(x) = \max_{i=1 \dots m} f(x, \zeta_i(\mathcal{I})) \quad (6.2)$$

if f is a function to minimize or

$$f^r(x) = \min_{i=1 \dots m} f(x, \zeta_i(\mathcal{I})) \quad (6.3)$$

if f is a function to maximize.

Two examples of an evaluation function will be presented in sections 6.3.2 and 6.4. Both will be the same type as function (6.1).

6.2.2. An example using a genetic algorithm

For clarity, we will use a genetic algorithm as a reference metaheuristic. The pseudo-code of this GA is given in algorithm 6.1.

Algorithm 6.1 Incremental genetic algorithm

- 1: *generate* an initial population
 - 2: **while** stopping conditions are not satisfied **do**
 - 3: *select* two individuals
 - 4: *crossover*: apply the crossover operator to the two individuals
 - 5: *mutation*: mutate offspring under probability
 - 6: *evaluation*: evaluate offspring
 - 7: *insertion* of offspring into the population under conditions
 - 8: *removal* of an individual from the population under conditions
 - 9: **end while**
 - 10: *report* the results
-

A large part of the initial population is randomly generated. Adding good quality solutions however, usually leads to faster convergence. These initial solutions are obtained most often through heuristics, resulting from previous studies. The stopping condition of the genetic algorithm is usually either a finite number of iterations, a finite number of iterations with no improvement of the best solution found, or a maximum computation time. On line 3 of algorithm 6.1, the choice of individuals is made either through binary tournament or by Reeves' ranking method [REE 95]. In both cases, it is the objective function value that determines the choice of the individuals. Mutation is carried out with a certain probability P_m . An important remark is that the crossover and mutation operators must be suitable for the problem [POR 96].

By following the two principles stated in section 6.2.1, the ordinary evaluation function is replaced by a *robust evaluation function* adapted to the problem. The insertion and removal of individuals also depends on their evaluation values. For example, we choose to insert a new individual if it is better than at least the worst individual of the population. Since the population size is constant, we remove an individual each time we insert one. The individual to be removed is an individual whose evaluation is not as good as the individual average evaluation and is selected through reverse binary tournament selection, for example. In several of these steps, the functioning of the genetic algorithm will be determined by the results of the robust evaluation through iterations and will thus lead to a robust solution.

Among similar studies, the genetic algorithm by Leon *et al.* [LEO 94] which solves a robust job-shop problem can be noted. This algorithm uses an objective function including robust measures. In this article the authors suppose that the machines can breakdown, but that operations immediately resume after the breakdown is finished (*right-shift reactive policy*). Robustness is defined as a schedule delay function, for example the difference between an effective makespan and an expected makespan (without breakdowns).

Another major piece of work in this domain is Jensen's doctoral thesis [JEN 01] which presents several innovative ideas. For example, he defines *neighborhood-based robustness*. The main idea is to select a partial set of high-quality solutions and to try to preserve them throughout the optimization. Their quality should not be altered even during disruptions. A particular genetic algorithm based on co-evolution is associated with this idea. *Co-evolution* is a term used to describe a type of genetic algorithm in which a population of robust solutions and a population of machine breakdowns evolve simultaneously. In the end, the former population consists of the most robust solutions and the latter of the worst machine breakdowns.

Other robust genetic algorithm references (Branke [BRA 98, BRA 01], Tsutsui [TSU 99] and Tsutsui and Gosh [TSU 97]) present studies on robust genetic algorithms but the applications used are limited to continuous mathematical function optimization. The work that is without a doubt the closest to ours is that presented in [FLE 04]. In this article, the authors present a precise stochastic evaluation of their objective function and give robust solutions for stochastic arc routing problems. To be able to compute a stochastic evaluation of a solution, the authors use in-depth statistical knowledge of the problem data.

6.3. Single-machine scheduling application

6.3.1. Minimizing the number of late jobs on a single machine

A set of jobs must be sequenced on a single machine which does not allow preemption. Each job is not delivered or available until its *release date*. The *processing time*, *due date*, as well as *weight* depending on the priority of each job, is assumed to be known. A job is considered *late* if its execution time overruns its due date. Otherwise, it is considered on time or *early*.

NOTE 6.1.– *Late jobs can be randomly placed after all of the early jobs without changing the objective value. In practice, one could even avoid sequencing them.*

The objective of the problem is to find a schedule which minimizes the weighted number of late jobs. Table 6.1 summarizes the notations used here.

Description	Notation	Comments
Number of jobs	n	
Release date	r_j	
Processing time	p_j	
Due date	d_j	
Weight	w_j	
Beginning time of processing	t_j	$r_j \leq t_j$
End time of processing	C_j	$C_j = t_j + p_j$
Late status	U_j	$U_j = 1$ iff $C_j > d_j$ and 0 otherwise

Table 6.1. Notations of a single-machine scheduling problem

In the standard classification, the problem is noted $1|r_j| \sum U_j$. This problem is \mathcal{NP} -hard in the strong sense [LEN 77]. In the deterministic case, a certain number of algorithms are able to efficiently solve this problem [DAU 95, BAP 99, DAU 03, SEV 03].

In this chapter, a genetic algorithm is used to solve the problem, first in the static case, then in the stochastic case. In this genetic algorithm, the solution is encoded as a permutation of n jobs. An initial population is randomly generated and the stopping condition is set to a finite number of iterations without an improvement of the best solution. The selection of individuals is carried out with Reeves' ranking method [REE 95] which consists of giving a greater probability of best individuals being selected. The crossover operator is a one-point crossover operator $X1$ [POR 96] and the mutation operator is the *general pairwise interchange GPI* which consists of exchanging two randomly selected jobs from the permutation. The mutation probability is set to $P_m = 0.25$. The evaluation is made by taking the jobs in the order of the permutation and by sequencing them straight away without changing the partial order. If a job cannot be placed *early*, by adhering to note 6.1, it can be pushed to the end of the schedule and its associated weight can be added to the final value of the objective function. Insertion and removal are carried out according to the instructions of section 6.2.2. In [SEV 03], several variations are studied with, notably, the addition of a local search which allows the best solutions to be found.

Even if all the parameters of our problem can be rendered stochastic, only the particular case in which the availability dates are deterministic will be modeled in this chapter.

6.3.2. Uncertainty of deliveries

A *just-in-time* production environment can be modeled like a scheduling problem in which the production order is represented as a series of jobs sequenced on a single machine. The solution to such a just-in-time problem consists of finding the sequence which minimizes the weighted number of late jobs. Other objectives can also be used, such as total weighted tardiness.

6.3.2.1. Considered problem

In just-in-time assembly, a large number of basic tasks can be regrouped into a reduced number of jobs. This number n varies between 20 and 80 each day. The horizon (a day) is divided into 80 five-minute periods. Earlier observations have shown that for a significant number of jobs, the availability dates promised by the suppliers for some parts or raw material are rarely respected causing a gap in the processing of associated jobs. The percentage of effected jobs is about 20% each day and the gap can go up to 20 units of time, about 1 hour and 40 minutes at the worst. Table 6.2 sums up the parameters of the problem generator used for this experiment. For each problem size (n), 20 instances are generated. The availability dates are generated according to a Gamma distribution giving a greater probability of appearing at the beginning of the horizon and the due dates are also generated according to a Gamma law but starting from the end of the horizon, giving a greater probability of appearing at the end of it.

Parameters	Values
Problem size	(20,40,60,80,100)
Horizon T	80
Availability date r_j	$\Gamma(0.2, 4)$
Due date d_j	$T - \Gamma(0.2, 4)$
Processing time p_j	$\mathcal{U}(1, d_j - r_j)$
Weight w_j	$\mathcal{U}(1, 10)$

Table 6.2. Rules of the instance generator

6.3.2.2. Robust evaluation function

The robust evaluation function evaluates each generated solution a fixed number (m) of times on the instance of the problem, the data of which has been modified. For each evaluation, a number of jobs are randomly selected and the availability dates of these jobs are increased by δ (defined in Table 6.3). The m evaluations are accumulated and the average is computed to determine the robust evaluation value. The robust evaluation parameters are summarized in Table 6.3. 20% of the jobs are delivered late and the lateness follows a uniform law between 0 and 20 units of time.

Parameters	Values
Percentage of jobs delivered late	20%
Observed lateness δ	$\mathcal{U}(0, 20)$
Number of evaluations (m) for f^r	100

Table 6.3. Robust evaluation parameters

The number of evaluations m must be high enough to avoid keeping any solution that is not robust. Moreover, a computed average in this case would have little meaning on a reduced number of evaluations. Value $m = 100$ from experience gives the adequate results, maintaining at the same time a reasonable total execution time.

6.3.3. Results

The results of the method can be evaluated by comparing the sequence given by the standard genetic algorithm (guided by the ordinary evaluation function, f) and the sequence given by the robust genetic algorithm (guided by the robust evaluation function f^r). The first method will be called SGA and the second RGA. The SGA results are noted first. Then, for the final sequence (corresponding to the best SGA result), we disrupt the data and measure the objective function value as well as the deviation between the two values.

Instance name	CPU (sec)	Nr. Iter.	Fitness f	Avg. f Pop	Avg. 1000r	Inc. (%)
ODD80_1	0.87	16775	400	405.05	430.17	7.54
ODD80_2	1.24	31903	349	353.90	373.00	6.88
ODD80_3	0.64	15606	348	354.19	371.25	6.68
ODD80_4	0.64	14505	411	417.14	431.00	4.87
ODD80_5	1.29	25518	307	312.76	337.28	9.86
ODD80_6	0.52	13217	329	332.62	340.99	3.64
ODD80_7	0.96	20278	331	336.86	361.88	9.33
ODD80_8	0.70	12646	354	357.67	368.84	4.19
ODD80_9	0.99	18863	317	321.43	343.06	8.22
ODD80_10	0.69	14645	344	347.67	366.58	6.56
ODD80_11	0.73	17557	394	398.19	417.80	6.04
ODD80_12	0.75	15224	363	374.52	385.68	6.25
ODD80_13	0.55	10500	317	322.81	338.57	6.81
ODD80_14	0.54	10104	364	368.86	389.80	7.09
ODD80_15	0.65	11040	369	373.86	385.32	4.42
ODD80_16	0.52	10418	370	375.90	384.52	3.92
ODD80_17	0.52	10982	325	331.48	342.11	5.26
ODD80_18	0.84	13599	307	312.81	324.60	5.73
ODD80_19	0.76	15434	357	363.43	376.29	5.40
ODD80_20	0.63	14365	365	372.33	391.88	7.36

Table 6.4. SGA results for the instances of 80 jobs

In Table 6.4, the instances of 80 jobs are analyzed. For each of the 20 instances, the SGA is run under the previously defined conditions. The CPU time and the number of iterations are measured. The fitness f – the value of the best solution function at the end of SGA – is given as well as the average fitness of the individuals of the population (column “Avg. f Pop”). Column “Avg. 1000r” gives, for the sequence at the end of the SGA run, the average evaluation of 1000 solutions, the data of which have been disrupted. For instance ODD80_1 for example, the best solution found has an objective function value of 406, whereas in disrupted mode the average value is noted at 430.17. This corresponds to an increase in relation to the expected solution of 7.54% (column “Inc.”).

Instance name	CPU (sec)	Nr. Iter.	Fitness		Avg. Pop	Avg. 1000r	Inc. (%)
			f	f^r			
ODD80_1	158.33	22058	406	406.36	411.76	417.60	2.86
ODD80_2	72.52	9956	357	358.91	361.90	366.73	2.72
ODD80_3	108.76	15687	369	363.37	365.62	374.23	1.42
ODD80_4	152.49	20818	429	422.61	424.57	426.65	-0.55
ODD80_5	224.35	29705	310	314.94	317.48	328.88	6.09
ODD80_6	100.98	15109	327	328.24	335.24	335.49	2.60
ODD80_7	147.44	20249	342	345.39	344.33	358.54	4.83
ODD80_8	96.03	11853	358	360.67	363.67	365.09	1.98
ODD80_9	120.26	14271	325	327.12	330.67	337.71	3.91
ODD80_10	126.65	16910	353	355.95	359.57	358.47	1.55
ODD80_11	119.67	15989	403	408.82	414.05	419.97	4.21
ODD80_12	94.76	13801	369	371.24	375.29	377.26	2.24
ODD80_13	62.12	8559	341	333.90	337.33	340.49	-0.15
ODD80_14	85.73	12212	381	378.84	379.76	383.52	0.66
ODD80_15	146.25	16869	373	377.04	382.81	378.48	1.47
ODD80_16	106.09	14472	377	370.44	377.38	380.52	0.93
ODD80_17	144.83	19637	329	329.30	332.00	333.23	1.28
ODD80_18	120.86	15289	313	316.72	315.86	321.17	2.61
ODD80_19	158.79	21159	364	363.40	363.81	368.20	1.16
ODD80_20	151.64	23057	370	371.78	376.43	377.56	2.04

Table 6.5. RGA results for the instances of 80 jobs

In Table 6.5, the same 80 jobs are analyzed after running the RGA. For the first instance, the best solution found had an objective value of 406 and in disrupted mode the average of 1000 modified instances is 417.60 which corresponds to an increase of 2.86%. Likewise, using the RGA sequence gives a certain advantage because in disrupted mode the average value is 417.60 compared to 430.17 for the SGA. However, this improvement comes at the cost of an increase in computing time. For the first instance, we go from less than one second for the SGA to more than 150 seconds.

Number of jobs	Difference		CPU time (s)	
	SGA (%)	RGA (%)	SGA	RGA
20	11.95	3.84	0.03	1.73
40	8.47	2.89	0.13	10.27
60	7.35	3.08	0.36	45.81
80	6.30	2.19	0.75	124.93
100	5.32	2.01	1.53	215.69

Table 6.6. Difference between the solution in the disrupted environment and in the non-disrupted environment

Table 6.6 shows the results for all of the instances. The use of the RGA gives less of a difference in disrupted mode and allows the construction of a sequence which will increase the objective function value by only a small amount. Our results confirm that taking the uncertain nature of a problem into account within the solving method is always beneficial, although an increase in computing time has to be taken into account.

6.4. Application to the planning of maintenance tasks

The SNCF or National Railway of France – the number one passenger transporter in that country – has a rolling stock of about 250 TGV high-speed trains which, for security reasons, need extremely regular maintenance. With this in mind, regional factories have specialized in maintenance, exemplified by the Hellemmes factory. These TGVs must be periodically serviced so that they can be used for the amount of time they were designed to last: 30 years. The mid-life maintenance operation is the most important one (with more than 10,000 basic operations). Our study focuses on this maintenance operation, which was the main task of the Hellemmes factory in 2000 and 2001.

With the increase in passenger traffic, the demand for trains has become greater in the last ten years. The SNCF therefore needs more and more simultaneously available trains to reach their annually set transport objectives. The cost of immobilizing a TGV for a day is not information that is available to the public, but can be considered as very high. In the last decade, the immobilization time of a TGV for the mid-life maintenance operation has been considerably reduced. Today, for a standard TGV (eight coaches), such as Paris-South-East TGV, the mid-life maintenance operation takes less than 40 days.

6.4.1. SNCF maintenance problem

The TGV maintenance problem is extremely complex and consists of several thousand basic tasks. For medium or short-term planning, TGV maintenance can be broken down into eight main tasks per coach. A TGV usually has eight coaches, giving us a total of 64 aggregated tasks which represent the principle stages of the TGV mid-life maintenance operation. The initial objective is to carry out all of the pre-established maintenance tasks while minimizing the total immobilization time. This problem is linked to a theoretical RCPSP multi-resource problem.

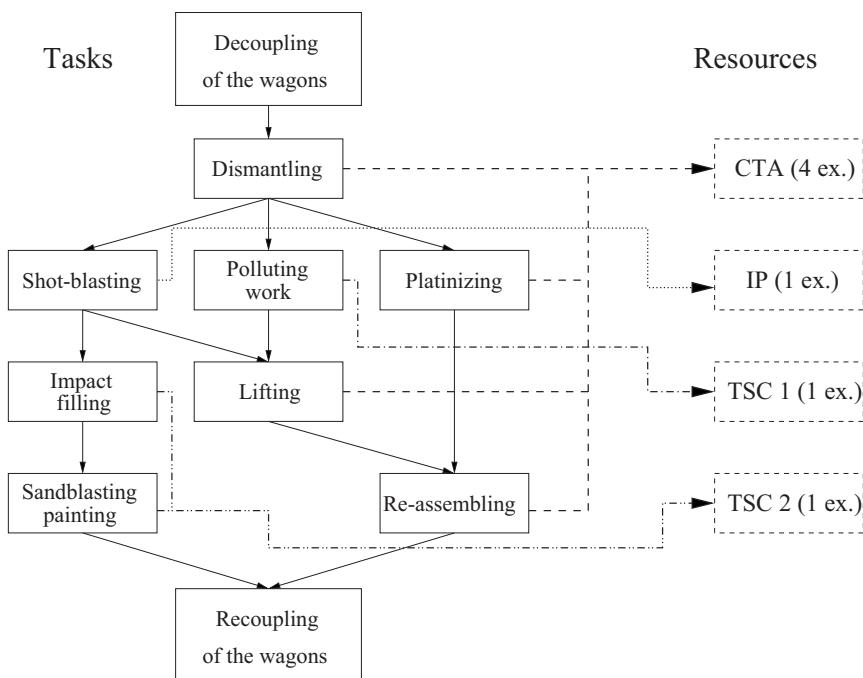


Figure 6.1. Precedence constraints and resource allocation

Figure 6.1 presents the different tasks for each coach, the precedence constraints between tasks, and the known resource allocation. The tasks related to the train (decoupling and recoupling) are not taken into account in our problem which deals only with the coaches. We can note that the CTA (Caisse TGV et Automoteur – TGV body and engine) is divided into four parts. CTA1 deals with the first two coaches, CTA2 with the next two, etc. There are two TSC (Traitement et Structure de Caisse – treatment and structure of the train's body) units, but they deal with different

operations. IP (Industries Privées – private industries) regroup the whole of the subcontractors and exclusively take care of the sand blasting of the coach structure.

	Task	Length
1	Dismantling	24
2	Sand blasting	15
3	Platinizing	8
4	Polluting work	10
5	Impact filling	8
6	Lifting	24
7	Sanding, painting	20
8	Re-assembling	24

Table 6.7. Length of tasks (in hours)

Table 6.7 shows the processing time of each task. The objective of our problem is to put the train back into commercial service for use as rapidly as possible which can be translated as a scheduling problem to minimize the makespan (C_{\max}). Ideally, the length of the tasks are known and, especially, set so that this problem can be formulated as a resource constrained scheduling problem [LE 01, LE 02]. Since the mid-life overhaul is a complex task, including the dismantling of the entire TGV, numerous unexpected issues can turn up. For example, during the dismantling of a floor, one might find that the support of the very floor needs to be changed. This implies that it is imperative all the unexpected operations to be done before reassembling. The goal of this study is therefore to propose a schedule that takes these unexpected events into account.

6.4.2. Uncertainties of an operational factory

In a factory of that size, unexpected factors often turn up, thus the production management service has a crucial role to play. First, a list of all the different unexpected factors encountered during mid-life maintenance operations on a Paris-South-East TGV must be drawn up over a period of 12 months.

Type of disruption	Frequency (in %)	Impact (in days)
Wrong diagnosis	30%	8
Logistics problem	11%	1
Bad coordination	11%	0
Workload change (external cause)	10%	1
Bad preparation of task	10%	3
Tool problem	10%	3
Supply problem	10%	1
Quality control failure	5.5%	7
Strike	2.5%	1

Table 6.8. Disruptions encountered by the SNCF

Table 6.8 shows the frequency of unexpected disruptions and more specifically their impact, i.e. the number of days delay in temporary planning. As can be seen from the table, incorrect diagnoses are the most frequent disruptions, making their impact on the schedule the greatest.

In our experiments, we create a robust schedule by allowing the duration of the tasks to be increased with time, a factor δ . The percentage of disrupted tasks is denoted pp .

6.4.3. A robust schedule

Once again, we use a genetic algorithm to solve the problem both in the static and disrupted case. The encoding used is a permutation of 64 jobs which correspond to the eight tasks of the eight coaches of a TGV. The availability of different resources is taken into account as follows. We use the permutation order as a priority order. A scheduling algorithm without delay is then used, and that attempts to place all jobs which could be processed in t with the resources if they are available. Then, we increment t and start the process again until all the jobs are placed. The choice of this technique is not insignificant, because it partially corresponds to the planning process used today.

As with the previous problem (section 6.3) we will create two evaluation functions, one *ordinary* and the other *robust* which are integrated into the genetic algorithm. The ordinary evaluation function simply consists of measuring the makespan after the placement of the scheduling algorithm without delay. For the robust evaluation function, we measure the different makespans obtained for the same sequence when the length of pp percentage of jobs are increased by a factor δ . The average of the different measures is the robust evaluation function. At the end of the process, we will have two sequences given by the SGA (ordinary evaluation function) and the RGA (robust evaluation function).

To be able to solve it, we looked into the different techniques available. As is usually the case, the manual method of solving the problem which is based on the experience of one man (henceforth called the *expert*), which takes several years to acquire, is the method currently used in the Hellemmes plant.

The choice of parameters for the disrupted case is not an easy one. In accordance with the SNCF experts, we can consider that an increase in the length of tasks can cover the all the unexpected factors in Table 6.8. Observations show that about 30% of the tasks are subject to an increase in processing time. For numerical experiments, the chosen parameters are thus $pp = 30\%$ and $\delta = 10$.

We use the three methods (expert, SGA and RGA) to find a first maintenance plan/schedule. Then, by preserving the obtained sequences, the disruptions are simulated for 1000 modified instances in which the length of the tasks increase by δ . As a result, the completion times of the different task are pushed back, in turn pushing back the makespan of the schedule. An average evaluation of the 1000 instances is shown in Table 6.9 (column Avg. 1000r). The table shows the results in hours.

Method	Time	Fitness		Avg.
used	CPU	f	f^r	1000r
expert	$\approx 1/2$ day	248	—	315.55
SGA	10.92s	228	—	295.65
RGA	479.13s	232	284.52	288.84

Table 6.9. Results (in hours) in the case of disruptions

Method used	Time	Fitness		Avg.
	CPU	f	f^r	1000r
expert	$\approx 1/2$ day	31	—	39.4
SGA	10.92s	28.5	—	37.0
RGA	479.13s	29	35.8	36.1

Table 6.10. Results (in days) in the case of disruptions

Table 6.10 shows the same results but this time the length is expressed in days.

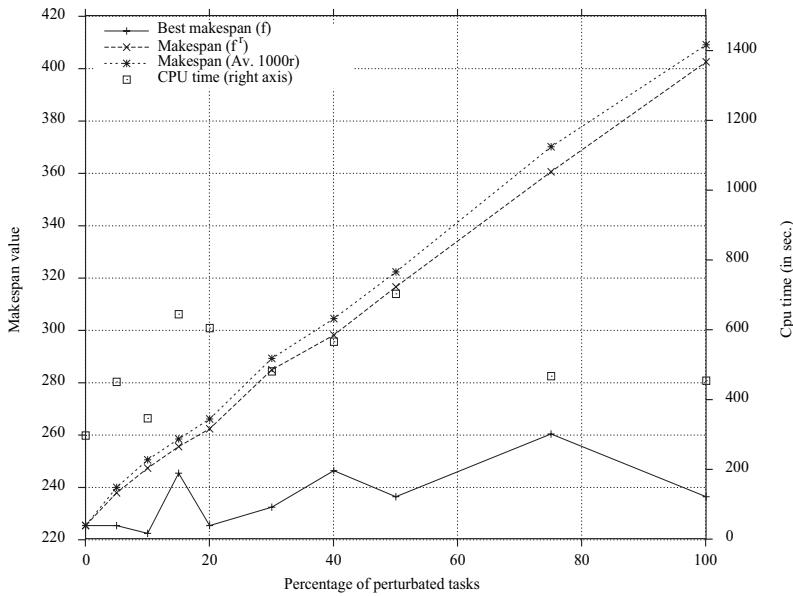
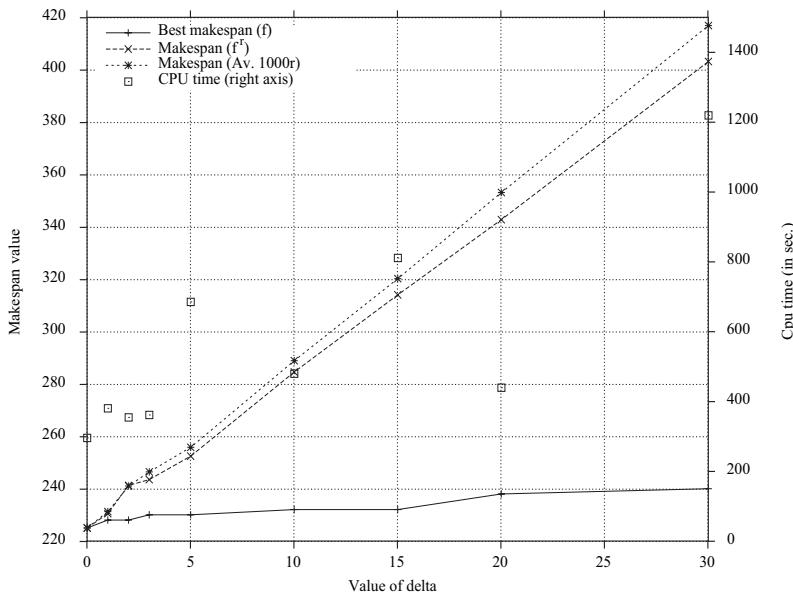
We can observe that, although the manual method is still the one used today, it is not competitive. In disrupted mode, the makespan is more than 39 days. However, the SGA and RGA give good results (37 and 36.1 days). It should be noted that only the RGA method can produce a schedule that is robust and has a high evaluation function value in case of disruption (column f^r).

6.4.3.1. Variations of the unexpected factors

The unexpected factors retained here are controlled by two parameters: pp , the number of jobs whose lengths are effected by the disruptions and δ , the value of the increasing in task length. To verify the accuracy of the method, we measured the makespan (f^r and Avg. 1000 r) when pp or δ varied. In Figures 6.2 and 6.3, the abscissa represents the two parameters analyzed, the left vertical axis, the value of the makespans and the right vertical axis, the CPU time in seconds.

Figure 6.2 shows the variation of the different makespan values when pp varies, value δ is set at 10. The percentage of disturbed tasks varies from 0 to 100%. The best makespan value varies between 225 and 260 hours. No link between pp and the value has been found. However, when pp increases, the gap between the temporary makespan and the post-computed makespan stays approximatively the same. This means that the proposed method is reliable for this case study.

In Figure 6.3 the percentage of disrupted tasks is set to $pp = 30\%$. The δ value varies in the interval [0, 30] interval (in hours). Examining the results, we reach the same conclusions as before. The gap between p-Makespan and e-Makespan is reduced, indicating that the method is reliable. A linear link between the variation of δ and the p-Makespan and e-Makespan values seems to exist. This can be explained by the method applied, which consists of simply pushing back the tasks to take the lateness into account, without questioning the order of the tasks.

**Figure 6.2.** Impact of the variation of pp **Figure 6.3.** Impact of the variation of δ

6.5. Conclusions and perspectives

The general robust optimization framework proposed in this chapter is a proactive approach which allows us to propose robust solutions to a scheduling problem and an SNCF maintenance task planning problem. The first point that we can highlight in this study is the simplicity of the use of the general framework. A metaheuristic can be adapted easily to a problem with stochastic data by modifying only one function.

A question that raises itself is how the general framework should be used? To answer this question, a decision-maker must take the following steps:

- 1) Determine the stochastic data of the problem and if possible model them. According to the problem, determine which robust evaluation function is the most adapted.
- 2) Use the general robust optimization framework to determine robust solutions to the problem.
- 3) Choose a proposed solution by answering the question “what would happen if?” by varying the different parameters related to the uncertain nature of the problem.

To help the decision-maker during the final choice of the robust solution, we can go back to the results presented in [SEV 04]. In this article we proposed measuring the distance of a set of solutions from the initial solution. We can measure the distance between the set of solutions found by the robust method and the basic solution (given by an expert, a previous solution or by another method on static data). With classic decision-making techniques, the decision-maker can then choose a typical interval for the robust evaluation function and choose a solution least sensitive to variations, while losing a little quality.

One of the next steps of our work will be research in robust evaluation functions in which the “simulation” part will be replaced by complete statistical evaluation. Such a function will allow us to noticeably reduce the computation time of the proposed approach (as in [FLE 04]).

For both applications, it would be interesting to complete this unique pro-active approach by a reactive approach, which would improve the understanding of the difficulties caused by data variations. For scheduling applications, the study could use the other results in this book. However, for the SNCF problem, other factors should be taken into account. For example, work has to be carried out in accordance with the SNCF representatives, who only rarely accept scheduling changes. Their hesitation most often brings about the transmission of wrong information making scheduling

updates almost impossible. Nowadays, questioning an existing schedule has become more exceptional, even if several tasks are already late during processing. Yann Le Quéré's thesis, to a certain extent, deals with the coordination of different decision centers and the integration of communication time in the re-scheduling process. This integration allows us to measure both the reactivity of a decision structure and the reactivity of the method used.

6.6. Bibliography

- [BAP 99] BAPTISTE PH., “An $\mathcal{O}(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs”, *Operations Research Letters*, vol. 24, p. 175–180, 1999.
- [BRA 98] BRANKE J., “Creating robust solutions by means of evolutionary algorithms”, *Parallel Problem Solving from Nature V*, LNCS vol. 1498, Springer Verlag, p. 119–128, 1998.
- [BRA 01] BRANKE J., “Reducing the sampling variance when searching for robust solutions”, SPECTOR L. and GOODMAN E. (Eds.), *GECCO 2001 – Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann Publishers, p. 235–242, 2001.
- [DAU 95] DAUZÈRE-PÉRÈS S., “Minimizing late jobs in the general one machine scheduling problem”, *European Journal of Operational Research*, vol. 81, p. 134–142, 1995.
- [DAU 03] DAUZÈRE-PÉRÈS S. and SEVAUX M., “Using Lagrangian relaxation to minimize the weighted number of late jobs”, *Naval Research Logistics*, vol. 50, no. 3, p. 273–288, 2003.
- [FLE 04] FLEURY G., LACOMME P. and PRINS C., “Evolutionary algorithms for stochastic arc routing problems”, *Applications of Evolutionary Computing. Proceedings of Evoworkshops*, LNCS 3005, p. 501–512, 2004.
- [HAO 99] HAO J.-K., GALINIER P. and HABIB M., “Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes”, *Revue d'intelligence artificielle*, vol. 13, no. 2, p. 283–324, 1999.
- [JEN 01] JENSEN M. T., Robust and flexible scheduling with evolutionary computation, PhD Thesis, University of Aarhus, Dept. of Computer Science, Denmark, October 2001.
- [KOU 97] KOUVELIS P. and YU G., *Robust Discrete Optimisation and its Applications*, vol. 14 of *Non-convex Optimization and its Applications*, Kluwer Academic Publishers, Dordrecht, 1997.
- [LE 01] LE QUÉRÉ Y., SEVAUX M., TRENTESAUX D. and TAHON C., “Planification réactive des opérations de maintien et d'actualisation réglementaire et technologique des systèmes complexes”, *Proceedings of the International Conference on Computer Aided Maintenance*, ENIM, Rabat, Morocco, p. A15/1–12, 2001.

- [LE 02] LE QUÉRÉ Y., SEVAUX M., TRENTESAUX D. and TAHON C., “Résolution d’un problème industriel de maintenance des TGV à la SNCF”, *4ième conférence nationale de la société française de recherche opérationnelle, ROADEF’2002*, Paris, France, 2002.
- [LEN 77] LENSTRA J. K., RINNOOY KAN A. H. G. and BRUCKER P., “Complexity of machine scheduling problems”, *Annals of Discrete Mathematics*, vol. 1, p. 343–362, 1977.
- [LEO 94] LEON V.J., WU S.D. and STORER R.H., “Robustness measures and robust scheduling for job shops”, *IIE Transactions*, vol. 26, p. 32–43, September 1994.
- [POR 96] PORTMANN M.-C., “Genetic algorithms and scheduling: a state of the art and some propositions”, *Proceedings of the Workshop on Production Planning and Control*, Mons, Belgium, p. I-XIV, 1996.
- [REE 95] REEVES C.R., “A genetic algorithm for flowshop sequencing”, *Computers and Operations Research*, vol. 22, no. 1, p. 5–13, 1995.
- [REE 97] REEVES C.R., “Genetic algorithms for the operations researcher”, *Informs Journal on Computing*, vol. 9, no. 3, p. 231–250, 1997.
- [SEV 03] SEVAUX M. and DAUZÈRE-PÉRÈS S., “Genetic algorithms to minimize the weighted number of late jobs on a single machine”, *European Journal of Operational Research*, vol. 151, no. 2, p. 296–306, 2003.
- [SEV 04] SEVAUX M. and SØRENSEN K., “A genetic algorithm for robust schedules in a one-machine environment with ready times and due dates”, *4OR*, vol. 2, no. 2, p. 129–147, 2004.
- [SØR 01] SØRENSEN K., “Tabu searching for robust solutions”, *Proceedings of the 4th Metaheuristics International Conference*, Porto, Portugal, p. 707–712, 2001.
- [SØR 03] SØRENSEN K., A framework for robust and flexible optimisation using metaheuristics – with applications in supply chain design, PhD Thesis, University of Antwerp, Belgium, 2003.
- [TSU 97] TSUTSUI S. and GHOSH A., “Genetic algorithms with a robust solution searching scheme”, *IEEE Transactions on Evolutionary Computation*, vol. 1, p. 201–208, 1997.
- [TSU 99] TSUTSUI S., “A comparative study on the effects of adding perturbations to phenotypic parameters in genetic algorithms with a robust solution searching scheme”, *Proceedings of the 1999 IEEE Systems, Man, and Cybernetics Conference (SMC’99 Tokyo)*, p. III-585–591, 1999.

Chapter 7

Metaheuristics and Performance Evaluation Models for the Stochastic Permutation Flow-Shop Scheduling Problem

In this chapter, we are interested in a basic scheduling problem: the permutation flow-shop. We are within the framework of a proactive approach. The uncertainty sources which we consider relate to the processing times of the jobs. These processing times are supposed to be uncertain (a function of the processing conditions (see Chapter 1)) and are described as random variables. The objective is to search for a robust solution which can be defined as a solution which minimizes a performance criterion z in expectation $E(z)$. As mentioned in Chapter 1, the objective is to provide a solution with a “good” behavior.

Our work falls within the framework of stochastic scheduling problems. Although this problematic is described as old, few results have been obtained in this domain to our knowledge. From a pure academic point of view, it is interesting to begin with the stochastic flow-shop, which is “largely” unstudied compared to the deterministic flow-shop, which has been the subject of many publications.

The outline of this chapter is as follows. In the next section, we define the stochastic permutation flow-shop scheduling problem under study. This presentation shows the existence of an underlying problem: a performance evaluation problem.

Chapter written by Michel GOURGAND, Nathalie GRANGEON and Sylvie NORRE.

This problem, the corresponding state of the art, and proposed models are considered in the second section. Section 7.3 deals with state-of-the-art methods proposed for the stochastic scheduling problem. The methods are implemented and tested on examples taken from the literature. Results are given in section 7.4.

7.1. Problem presentation

For several years, the static deterministic flow-shop scheduling problem has been extensively researched and continues to be the subject of many investigations. A deterministic permutation flow-shop (Figure 7.1) is composed of a set of m machines and n jobs. The n jobs are processed by the machines in the same order (machine M_1 , machine M_2, \dots , machine M_m). Each job T_j , $j = 1, \dots, n$ is processed during a time p_{ij} by each machine M_i , $i = 1, \dots, m$.

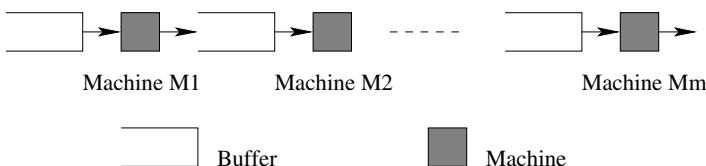


Figure 7.1. The flow-shop model

Traditional assumptions for the deterministic flow-shop are the following:

- H1** job release dates are known,
- H2** machines are always available,
- H3** processing times are deterministic and independent,
- H4** setup times and removal times are included in processing times,
- H5** transportation times are negligible,
- H6** there is no splitting,
- H7** a machine cannot process more than one job at a time,
- H8** no job may be processed by more than one machine at a time,
- H9** between two machines, jobs can wait in an unlimited buffer, managed according to the FIFO rule.

The deterministic flow-shop scheduling problem consists of finding a job schedule which minimizes a criterion. For instance, the criterion may be the completion time of the last job (makespan), the total flow time, the tardiness, etc.

In this chapter, we are interested in a stochastic flow-shop in which the processing times are modeled by random variables. Assumption H3 is replaced by the following assumption:

H3' The processing times are modeled by independent random variables.

In the literature, we mainly find the geometric, exponential, uniform and Erlang distributions.

The buffer before a machine can have any capacity (limited or unlimited) or be nonexistent. Assumption H9 is replaced by the following assumption:

H9' There is a FIFO buffer of capacity b_i ($b_i \geq 0$) before machine M_i ($i = 2, \dots, m$) and $b_1 = \infty$.

We consider the problem of finding a job schedule on the first machine which minimizes an expected criterion. By using the notation described in Chapter 1, the criterion of a schedule S is minimized in expectation if its expected criterion $E_{\mathcal{I} \in \mathcal{P}}(z_{\mathcal{I}}(S))$ is lower than the expected criterion $E_{\mathcal{I} \in \mathcal{P}}(z_{\mathcal{I}}(S'))$ of any other schedule S' . By using the notation proposed by [GOU 00], the problem can be noted:

$$\text{Fm } |b_i, p_{i,j} \sim \text{general}| E(z)$$

where b_i is the capacity of the buffer before machine M_i .

In the literature on the stochastic permutation flow-shop problem, other problems are considered:

- finding a job schedule which stochastically minimizes a performance criterion, noted z_{st} . The criterion of a schedule S is stochastically minimized, if it is stochastically lower than the criterion of any other schedule, i.e. $P(z(S) \leq t) \geq P(z(S') \leq t), \forall t, \forall S'$;

- finding a job schedule which maximizes the robustness, noted $\text{Rob}(z)$. In [KOU 00], the robustness of a schedule S in a two machine flow-shop is defined by the maximum deviation between the makespan of schedule S and the optimal makespan (computed by using the Johnson rule) for a set of scenarios of processing times.

Figure 7.2 lists the static scheduling problems studied in the literature. References are given in Table 7.1. A detailed state of the art for the stochastic flow-shop is delineated in [GOU 00] as well as a state of the art for the flow-shop with breakdowns. We will talk again about the state of art in the section about performance evaluation and in the section about scheduling.

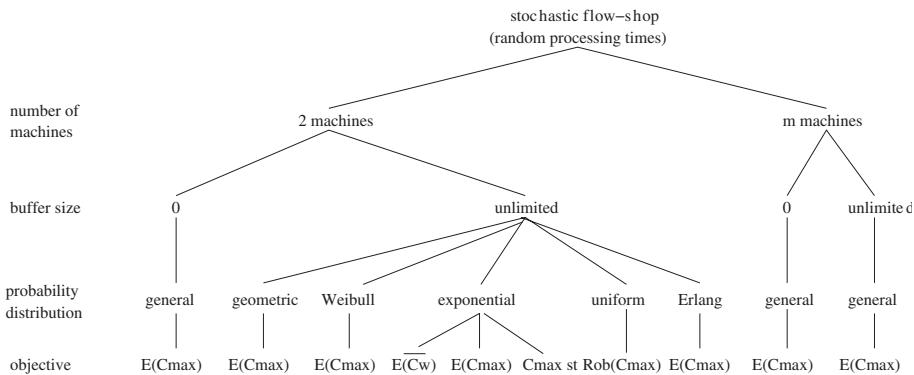


Figure 7.2. State of the art: flow-shop problem with random processing times

m	$b_{i,i+1}, \forall i = 1, \dots, m - 1$	Probability distribution	Objective	References
2	0	general	$E(C_{\max})$	[PIN 82, JIA 98]
2	∞	general	service level	[POR 06]
2	∞	geometric	$E(C_{\max})$	[PRA 81]
2	∞	exponential	$E(\bar{C}_w)$	[FOR 81, FOR 83]
2	∞	exponential	$E(C_{\max})$	[MAK 65, TAL 67, BAG 70b, CUN 73, MIT 77, WEI 82]
2	∞	Weibull	$E(C_{\max})$	[KAL 06]
2	∞	exponential	$C_{\max} st$	[KU 86, KAM 99]
2	∞	uniform	$Rob(C_{\max})$	[KOU 00]
2	∞	Erlang	$E(C_{\max})$	[BAG 70a]
m	0	general	$E(C_{\max})$	[BAG 70a, PIN 82, FOL 84]
m	∞	general	$E(C_{\max})$	[MAK 65, TAL 67, PIN 82]

Table 7.1. State of the art: flow-shop problem with random processing times

When random events disturb the system, an underlying problem exists: a performance evaluation problem. Indeed, in a framework where all the data are deterministic, the criterion of a schedule only depends on deterministic values. In a

stochastic model, the criterion depends on a great number of random values. In the next section, we propose models for performance evaluation: a Markovian model and a Monte Carlo simulation model. The first model allows us to compute the exact value of a criterion in expectation whereas the second one provides an estimation. Then, we propose methods for the scheduling problem and combinations (Figure 7.3) between a performance evaluation model and an optimization method (more often metaheuristics).

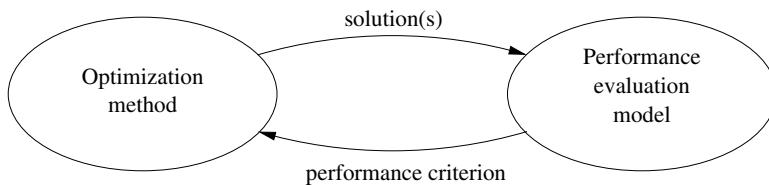


Figure 7.3. Combination between an optimization method and a performance evaluation model

7.2. Performance evaluation problem

Existing works about stochastic flow-shops propose methods for building optimal schedules but generally do not give any method for evaluating the performance criterion. Bagga, in [BAG 70a], expresses the expectation of the makespan by using n integrals, but does not give any computation method. From this observation, Cunningham and Dutta propose a method for computing the makespan expectation $E(C_{\max})$ for a two machine flow-shop with unlimited buffers and processing times exponentially distributed [CUN 73].

We propose to generalize this study in the following way: to solve the performance evaluation problem, we use two methods, the Markovian analysis and the Monte Carlo simulation.

7.2.1. Markovian analysis

A permutation flow-shop with buffers managed according to the FIFO rule is a queuing system, where n already-arrived customers are to be served by m service channels in series. When processing times follow exponential distributions with a known rate, this system can be modeled as a Markov chain. Solving this Markov chain (with QNAP2 software for example) gives the exact value of $E(C_{\max})$. In order

to avoid the use of a dedicated software, we have generalized the method proposed by [CUN 73] for the two machine flow-shop problem with unlimited buffer to the m machine flow-shop problem. This generalization will allow us to deal with larger size problems than with QNAP2, but it will also be limited in terms of number of jobs and number of machines (which is natural in Markovian analysis).

In this part, we propose to use Markovian analysis to compute the exact value of the expectation of the makespan for the problems:

$$\begin{aligned} \text{Fm } |p_{i,j} \sim \exp(\mu_{i,j})| E(C_{\max}) & \quad \text{unlimited buffers} \\ \text{Fm } |b_i = 0, p_{i,j} \sim \exp(\mu_{i,j})| E(C_{\max}) & \quad \text{no buffer} \\ \text{Fm } |b_i, p_{i,j} \sim \exp(\mu_{i,j})| E(C_{\max}) & \quad \text{limited buffers} \end{aligned}$$

In the following, we assume that the n jobs are numbered according to the lexicographical sequence $x = (1, 2, 3, \dots, j, j+1, \dots, n)$ and the processing time of job T_j , $j = 1, \dots, n$ by machine M_i , $i = 1, \dots, m$ follows an exponential distribution with rate $\mu_{i,j}$. Random variables are assumed to be independent.

A job can block a machine:

- when there is no buffer between two machines, the job completed by the first of the two machines blocks it if the second one is processing another job;
- when there is a limited buffer between two machines, the job completed by the first of the two machines blocks it if the buffer is full.

In order to use the Chapman-Kolmogorov equations, we need to represent the states of the system and all the previous states of a given state. A state of the system is represented by a m length vector:

$$\vec{k} = (k_1, k_2, \dots, k_m)$$

with the following relations:

$$n + 1 \geq |k_1| \geq \dots \geq |k_i| \geq |k_{i+1}| \geq \dots \geq |k_m| \geq 1 \quad (7.1)$$

$$0 \leq |k_i| - |k_{i+1}| \leq b_{i+1} + 1, \quad i = 1, \dots, m - 1 \quad (7.2)$$

$$|k_i - k_{i+1}| < 2|k_i|, \quad i = 1, \dots, m - 1 \quad (7.3)$$

$$k_i + k_{i+1} > -2|k_i|, \quad i = 1, \dots, m - 1 \quad (7.4)$$

and $b_i \geq 0$ is the capacity of the buffer before machine i ($i = 2, \dots, m$) ($b_1 = \infty$).

The absolute value $|k_i|$ represents:

– either a job number:

- the job number that is being processed by machine M_i (if $k_i \geq 1$),
- the job number that is blocked by machine M_i (if $k_i \leq -1$),
- the job number that machine M_i (if $k_i = k_{i-1} \geq 1$) waits for.

– or the fact that all the jobs have been processed by machine M_i , $i = 1, \dots, m$ ($k_i = n + 1$).

More precisely, $|k_i|$ represents a job:

- 1) If $k_1 \geq 1$ and $k_1 < n + 1$, then machine M_1 is processing job T_{k_1} .
- 2) If $k_i \geq 1$ and $k_i < k_{i-1}$, then machine M_i is processing job T_{k_i} , $i = 2, \dots, m$.
- 3) If $k_i \geq 1$ and $k_i = k_{i-1}$, then machine M_i is idle and is waiting for job T_{k_i} , $i = 2, \dots, m$ (T_{k_i} is the next job to be processed by machine M_i).
- 4) If $k_i \leq -1$, then job $T_{|k_i|}$ blocks machine M_i (the buffer of machine M_{i+1} is full ($b_{i+1} > 0$) or machine M_{i+1} is busy ($b_{i+1} = 0$)).

We consider two particular cases:

– $\overrightarrow{k^{(1)}} = (1, 1, \dots, 1)$ with $k_i = 1, \forall i = 1, \dots, m$ represents the initial state of the system. The job T_1 is processed by the first machine and the other machines are idle.

– $\overrightarrow{k^{(N)}} = (n + 1, n + 1, \dots, n + 1, n)$ with $k_i = n + 1, \forall i = 1, \dots, m - 1$ and $k_m = n$ represents the state where job T_n is processed by machine M_m and other machines are idle (they have processed all the jobs).

Let N be the number of states of the system.

In a closed queuing network composed of m stations and processing a constant number j of customer, the number of states is:

$$\binom{m-1}{j+m-1}$$

When the capacity of the buffers is unlimited, N is given by the formula:

$$N = \sum_{j=1}^n \binom{m-1}{j+m-1}$$

In a two machine flow-shop without buffer, N is given by formula:

$$N = 3n - 1$$

In a two machine flow-shop with a buffer of capacity b between the two machines, N is given by the formula:

$$N = (b + 3)n - (b + 1)(b + 2)/2$$

EXAMPLE.– Let there be a six machine flow-shop with 12 jobs. Table 7.2 gives the buffer capacity.

Machine	M_2	M_3	M_4	M_5	M_6
Buffer capacity	1	2	0	2	3

Table 7.2. Buffer capacity

The state $\vec{k} = (13, -12, 9, 8, 8, 5)$ is represented in Figure 7.4:

- $k_1 = 13$ means that all the jobs have been processed by machine M_1 .
- $k_2 = -12$ means that job T_{12} blocks machine M_2 .
- $k_3 = 9$ means that job T_9 is being processed by machine M_3 .
- $k_4 = k_5 = 8$ means that machine M_4 is processing job T_8 and machine M_5 is idle, waiting for job T_8 .
- $k_6 = 5$ means that machine M_6 is processing job T_5 .

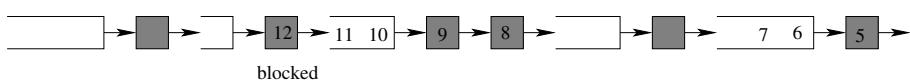


Figure 7.4. $\vec{k} = (13, -12, 9, 8, 8, 5)$

As the jobs are processed in lexicographical sequence, job numbers in the buffers can be deduced:

- jobs T_{10} and T_{11} wait in the buffer before M_3 ;
- jobs T_6 and T_7 wait in the buffer before M_6 .

To study the states preceding state \vec{k} , we introduce the vector $\vec{a}(\vec{k}, i)$ defined by:

- 1) $\vec{a}(\vec{k}, i) = (k_1, \dots, k_{i-1}, k_i - 1, k_{i+1}, \dots, k_m)$ if $(|k_{i-1}| < b_i + k_i + 1$ or $i = 1)$ and $(k_i > 0)$. Job T_{k_i-1} is processed by machine M_i .

2) $\vec{a}(\vec{k}, i) = (k_1, \dots, k_{i-1}, |k_i|, k_{i+1}, \dots, k_m)$ if $k_i < 0$. Job T_{k_i} is processed by machine M_i .

3) $\vec{a}(\vec{k}, i) = (k_1, \dots, -(k_{i'} - 1), \dots, -(k_{i-1} - 1), k_i - 1, k_{i+1}, \dots, k_m)$ in the other cases, where $i' = \max\{\max_{l=1, \dots, i-1}\{l/k_l < 0\}, \max_{l=2, \dots, i-1}\{l/|k_{l-1}| = b_l + k_l + 1\}, 1\}$ and $M_{i'}$ is:

- the first upper machine which is blocked,
- or the first upper machine with a non-full buffer,
- or by default the first machine of the flow-shop.

Job $T_{k_{i-1}}$ is processed by machine M_i and one or more jobs is blocked on the upper machines.

The set of vectors $\vec{a}(\vec{k}, i), \forall i = 1, \dots, m$ includes all the previous states of \vec{k} . This set may contain vectors with no significance for our problem: they do not verify relation (7.1), (7.2), (7.3) or (7.4).

EXAMPLE.—

$$\begin{aligned}
 \vec{k} &= (13, -12, 9, 8, 8, 5) \\
 \vec{a}(\vec{k}, 1) &= (12, -12, 9, 8, 8, 5) \quad (\text{case 1}) \\
 \vec{a}(\vec{k}, 2) &= (13, 12, 9, 8, 8, 5) \quad (\text{case 2}) \\
 \vec{a}(\vec{k}, 3) &= (13, 13, 8, 8, 8, 5) \quad (\text{case 3}) \\
 \vec{a}(\vec{k}, 4) &= (13, 13, -8, 7, 8, 5) \quad (\text{case 3}) \\
 \vec{a}(\vec{k}, 5) &= (13, -12, 9, 8, 7, 5) \quad (\text{case 1}) \\
 \vec{a}(\vec{k}, 6) &= (13, -12, 9, 8, 8, 4) \quad (\text{case 1})
 \end{aligned}$$

In this example, vectors $\vec{a}(\vec{k}, 2)$, $\vec{a}(\vec{k}, 5)$ and $\vec{a}(\vec{k}, 6)$ are previous states of state \vec{k} . They are represented by Figures 7.5, 7.6 and 7.7. Vectors $\vec{a}(\vec{k}, 1)$, $\vec{a}(\vec{k}, 3)$ and $\vec{a}(\vec{k}, 4)$ do not correspond to states of the system (relation (7.1), (7.2), (7.3) or (7.4) is not verified):

- In $\vec{a}(\vec{k}, 1)$, machine M_1 is processing job T_{12} , the next machine cannot be blocked by job T_{12} (relation (7.3) is not verified).
- In $\vec{a}(\vec{k}, 3)$, machine M_3 is processing job T_8 and jobs T_{12} , T_{11} , T_{10} and T_9 wait in the buffer before the machine. The buffer capacity is exceeded (relation (7.2) is not verified).
- In $\vec{a}(\vec{k}, 4)$, machine M_4 is processing job T_7 , but machine M_5 is processing job T_8 and the lexicographical sequence is not respected (relation (7.1) is not verified).



Figure 7.5. $\vec{a}(\vec{k}, 2) = (13, 12, 9, 8, 8, 5)$

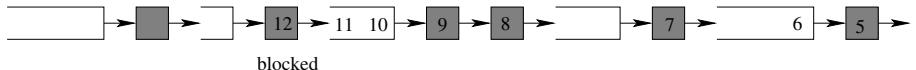


Figure 7.6. $\vec{a}(\vec{k}, 5) = (13, -12, 9, 8, 7, 5)$

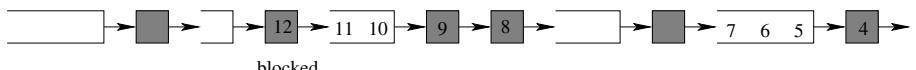


Figure 7.7. $\vec{a}(\vec{k}, 6) = (13, -12, 9, 8, 8, 4)$

Let

$$\tau(\vec{k}, i) = \begin{cases} 0 & \text{if } k_i = k_{i-1} \text{ or } k_i < 0 \text{ or } k_i = n+1 \\ & \text{or } \vec{k} \text{ is not a state of the system} \\ \mu_{k_i, i} & \text{otherwise} \end{cases}$$

$\tau(\vec{k}, i)$ is the processing rate of job T_{k_i} for machine M_i , if T_{k_i} is being processed by machine M_i . It is equal to zero in the following cases:

– \vec{k} does not represent a state of the system. Relation (7.1), (7.2), (7.3) or (7.4) is not verified;

– \vec{k} represents a state of the system but:

- if $k_i = k_{i-1}$, then machine M_i is not processing job T_{k_i} but is waiting for it,
- if $k_i < 0$, then job $T_{|k_i|}$ blocks machine M_i ,
- if $k_i = n+1$, then there is no job being processed by machine M_i as all jobs have already been processed.

With the notations, the following theorem is proposed and proved [GOU 05]:

THEOREM 7.1.– In a m -machine permutation flow-shop, under the following assumptions:

– there is a buffer of capacity b_i in front of machine M_i ($b_i \geq 0, \forall i = 2, \dots, m$, $b_1 = \infty$),

– the buffers are managed by the FIFO rule,

– the n jobs are processed according to the lexicographical sequence: $S = (1, 2, \dots, j, j + 1, \dots, n)$,

– the job processing times are independent and exponentially distributed: the processing time of job T_j , ($j = 1, \dots, n$) for machine M_i , ($i = 1, \dots, m$) follows an exponential distribution function with rate $\mu_{i,j}$.

The expected makespan $E(C_{\max})$ is computed using the formula:

$$E(C_{\max}) = -\mu_{n,m} \beta(\overrightarrow{k^{(N)}})$$

where

$$\alpha(\overrightarrow{k^{(1)}}) = 1/\mu_{1,1} \quad (7.5)$$

$$\beta(\overrightarrow{k^{(1)}}) = -1/\mu_{1,1}^2 \quad (7.6)$$

$$\alpha(\overrightarrow{k}) = \frac{\sum_{i=1}^m \alpha(\overrightarrow{a}(\overrightarrow{k}, i)) \tau(\overrightarrow{a}(\overrightarrow{k}, i), i)}{\sum_{i=1}^m \tau(\overrightarrow{k}, i)} \quad (7.7)$$

$$\beta(\overrightarrow{k}) = \frac{-\alpha(\overrightarrow{k}) + \sum_{i=1}^m \beta(\overrightarrow{a}(\overrightarrow{k}, i)) \tau(\overrightarrow{a}(\overrightarrow{k}, i), i)}{\sum_{i=1}^m \tau(\overrightarrow{k}, i)} \quad (7.8)$$

NOTE.– The expected makespan for any other sequence S' can be directly obtained from the expression of $E(C_{\max})$ simply by interchanging the appropriate subscripts.

Table 7.3 shows existing works concerning Markovian models for the stochastic flow-shop with exponentially distributed processing times.

flow-shop	unlimited buffer FIFO	limited buffer or no buffer FIFO
2 machines	[CUN 73]	[GOU 01]
m machines	[GOU 03]	[GOU 05]

Table 7.3. Existing works on the evaluation of $E(C_{\max})$ for a flow-shop with exponentially distributed processing times

7.2.2. Monte Carlo simulation

Markovian models allow us to compute the expected makespan when the processing times are modeled by random variables which follow rational Laplace transform distributions. For other cases (other criterion, other distribution, etc.), Markovian models cannot be used, so we use Monte Carlo simulation models to

evaluate $E_{\mathcal{I} \in \mathcal{P}}(Z_{\mathcal{I}}(S))$ and the corresponding confidence interval. This model has been validated by comparison with the results obtained with the queuing network model written in QNAP2 software [QNA 94] and solved by simulation. QNAP2 is a queuing network analysis package, version 2. QNAP2 is a performance evaluation software for systems modeled by queuing networks. Three types of solving methods are proposed by the software: analytical methods (exact and approached), Markovian analysis and discrete event simulation.

A flow-shop is modeled as a terminating system: initial state and final state correspond to the same state: no job is being processed. We can successively realize several statistically independent simulations, called replications. At each replication, a sample of job processing times is chosen and the corresponding criterion is computed. The mean of the obtained criteria is an estimation of the expected criterion. The number of replications must be large enough to assure a good sampling of the stochastic model behavior. The principle algorithm is given in Figure 7.8.

Let:

- $NbRep$: the number of replications,
- \mathcal{I}_r : a sample of processing times of each job for each machine according to the distribution functions ($r = 1, \dots, NbRep$),
- S : a schedule,
- $z_{\mathcal{I}_r}(S)$: the criterion corresponding to S according to \mathcal{I}_r ,
- $\overline{z(S)}$: an estimation of the expected criterion for S ,
- CI : the confidence interval of the estimation of the expected criterion.

```

for  $r = 1, \dots, NbRep$  do
    Choose randomly  $\mathcal{I}_r$ 
    Evaluate  $z_{\mathcal{I}_r}(S)$ 
end for
 $\overline{z(S)} := \sum_{r=1}^{NbRep} z_{\mathcal{I}_r}(S) / NbRep$ 
Compute the confidence interval  $CI$ 

```

Figure 7.8. Evaluation of an estimation of $E(Z(S))$

When the size of the problem allows it, the Markovian model is used to validate the simulation model (i.e. the exact value of Markovian model is in the confidence interval of simulation model).

7.3. Scheduling problem

The problem of minimizing of the performance criterion in expectation can be formulated as a combinatorial optimization problem:

$$\min_{S \in \Omega} E_{\mathcal{I} \in \mathcal{P}}(z_{\mathcal{I}}(S))$$

where Ω is the set of feasible solutions and z the criterion to minimize in expectation.

The first publication about flow-shop scheduling with random processing times appeared in 1965. Makino in [MAK 65] developed a sequencing rule to find the schedule that minimizes the expected makespan in a flow-shop with two machines, two jobs, unlimited buffers and exponentially distributed processing times. Talwar [TAL 67], in 1967, extended Makino's rule to the cases of three and four jobs and speculates upon its use for more than four jobs. In 1973 Cunningham and Dutta [CUN 73] proved Talwar's conjecture called, in the following text, "Talwar's rule".

THEOREM 7.2.— *Scheduling jobs in decreasing order of $\mu_{1,j} - \mu_{2,j}$ minimizes $E(C_{\max})$ in a two machine flow-shop with processing times exponentially distributed. ($\mu_{i,j}$ is the rate of the processing time of job T_j , $j = 1, \dots, n$ for machine M_i , $i = 1, 2$).*

State of the art resources about the stochastic flow-shop scheduling problem have been published by: Forst [FOR 84], Righter in Chapter 11 of [SHA 94], Pinedo [PIN 95] and Gourgand *et al.* [GOU 00]. In this latter reference, like many authors, we noted that the stochastic flow-shop problem was studied from 1965 to 1980, then forsaken by researchers and again studied since the late 1990s. Results presented in the literature correspond to particular cases with restrictive assumptions. Moreover, works mainly deal with the two machine flow-shop problem.

Metaheuristics are combinatorial optimization methods which allow us to deal with high combinatorial optimization problems with complex constraints such as technological constraints, routing constraints, etc. Metaheuristics are general principles and can be used for a large class of problems [HAO 99]. One of its main advantages lies in the possibility of controlling the processing time: the quality of the solution is progressively improved and the user can stop the execution at any time. The simulated annealing algorithm have been proved to converge in probability towards an optimal solution [KIR 83], [HAJ 88]. These methods are based on the concept of a neighboring system. The current solution S is modified by applying a

transformation to obtain a new solution S' , called a neighbor of S . The definition of this transformation generates a function which associates the set of neighbor solutions with each solution. This function is called a neighboring system.

The implementation of a metaheuristic (such as simulated annealing) ensures the value of the criterion is computed, at each iteration. To do this, we propose to combine a performance evaluation model and a metaheuristic. The main principle of the combination is described in [CAU 95].

7.3.1. Comparison of two schedules

When using an iterative improvement method, we must compare, at each iteration, two schedules S and S' . S is the current solution and S' is the candidate solution. If the Markovian model can be used, $E(z(S))$ and $E(z(S'))$ are exactly known and they can be compared without any problem. The schedule S' is better than the schedule S if $E(z(S')) \leq E(z(S))$.

Otherwise, we do not know the exact value of $E(z(S))$ and $E(z(S'))$ but only an estimation $\overline{z(S)}$ and $\overline{z(S')}$ obtained using the simulation model. In this case, the evaluations of $z_{\mathcal{I}_r}(S)$ and $z_{\mathcal{I}_r}(S')$ are computed according to the same samples. When the performance evaluation model is a discrete event simulation model, proposed algorithms compare schedules according to estimations of the expected performance criterion. As these estimations are means, we propose to extend the comparative process by using the confidence interval of the mean in the following way.

As $E(z(S))$ is unknown, we compute an estimation v_S^2 of the variance of $z(S)$:

$$v_S^2 = \frac{\sum_{r=1}^{NbRep} (z_{\mathcal{I}_r}(S) - \overline{z(S)})^2}{NbRep - 1}$$

v_S^2 being an estimator of the variance whatever the distribution function. The confidence interval of $\overline{z(S)}$ with 95% probability is:

$$\overline{z(S)} \pm 2\sigma_S / \sqrt{NbRep} \quad \text{with} \quad \sigma_S \simeq v_S$$

In the traditional or first version, the schedule S' , chosen in the neighborhood of S , is better than S if the mean $\overline{z(S')}$ is lower than the mean $\overline{z(S)}$:

$$S' \text{ is better than } S \quad \text{if } \overline{z(S')} \leq \overline{z(S)}$$

We propose to modify the acceptance criterion of a neighbor by using the confidence interval. Three other versions are proposed.

In the second version, the schedule S' , chosen in the neighborhood of S , is better than S if the mean $\overline{z(S')}$ is lower than the lower bound of the confidence interval of $\overline{z(S)}$:

$$S' \text{ is better than } S \text{ if } \overline{z(S')} < \overline{z(S)} - 2v_S / \sqrt{NbRep}$$

In the third version, the schedule S' , chosen in the neighborhood of S , is better than S if the upper bound of the confidence interval of $\overline{z(S')}$ is lower than the lower bound of the confidence interval of $\overline{z(S)}$:

$$S' \text{ is better than } S \text{ if } \overline{z(S')} + 2v_{S'} / \sqrt{NbRep} < \overline{z(S)} - 2v_S / \sqrt{NbRep}$$

In the fourth version, the schedule S' , chosen in the neighborhood of S , is better than S if the mean $\overline{z(S')}$ is lower than the mean $\overline{z(S)}$ and if the upper bound of the confidence interval of $\overline{z(S')}$ is lower than the upper bound of the confidence interval of $\overline{z(S)}$:

$$\begin{aligned} S' \text{ is better than } S \text{ if } \overline{z(S')} + 2v_{S'} / \sqrt{NbRep} &< \overline{z(S)} + 2v_S / \sqrt{NbRep} \\ \text{and } \overline{z(S')} &< \overline{z(S)} \end{aligned}$$

The four versions are illustrated in Figure 7.9.

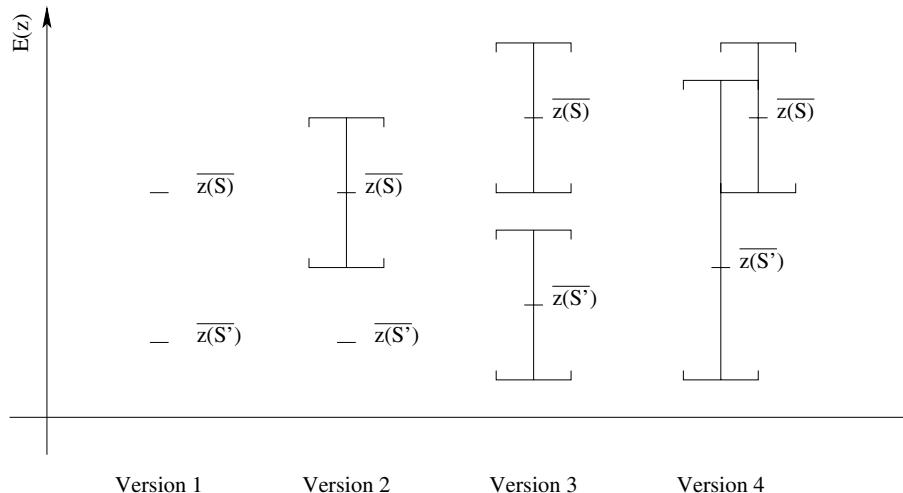
7.3.2. Stochastic descent for the minimization in expectation

The basic algorithm is the stochastic descent (Figure 7.10). The neighbor solution is accepted if it is better (according to section 7.3.1) than the current solution. This algorithm allows us to find generally a local minimum.

The initial solution can be randomly generated or obtained by a heuristic usually used for the deterministic model, for example. The algorithm stops after a given time or a given number of iterations.

7.3.3. Inhomogenous simulated annealing for the minimization in expectation

The simulated annealing method (Figure 7.11) was proposed in 1983 by Kirkpatrick [KIR 83] for solving combinatorial optimization problems. The simulated annealing is an extended version of stochastic descent. It looks for solutions

**Figure 7.9.** Comparison of two solutions S and S'

```

Let  $S$  be an initial solution,
while necessary do
    Choose  $S'$  randomly and uniformly in the neighborhood  $\mathcal{V}$  of  $S$ .
    if  $S'$  is better than  $S$  then
         $S := S'$ 
    end if
end while
 $S$  is the solution of the method
  
```

Figure 7.10. Principle algorithm of the stochastic descent

with low cost while accepting, in a controlled way, solutions which may degrade the objective function. At each iteration, a neighbor $S' \in \mathcal{V}(S)$ of the current solution S is randomly chosen. If S' is better than S , then the solution S' is systematically accepted. Otherwise, S' is accepted with a probability $p(\Delta z, T_k)$ which depends on the difference $\Delta z = E(z(S')) - E(z(S))$ (the small deteriorations are more easily accepted) and on the temperature T (a high temperature corresponds to a great probability to accept deteriorations). The temperature is controlled by a decreasing function which defines a cooling scheme.

```

Let  $S$  be an initial solution,  $T_0$  an initial temperature,
 $S_{\text{best}} := S$ 
while necessary do
    Choose  $S'$  randomly and uniformly in the neighborhood  $\mathcal{V}$  of  $S$ .
    if  $S'$  is better than  $S$  do
         $S := S'$ 
    else
        if  $\exp\left(\frac{\Delta z}{T_k}\right) > \text{random}[0, 1]$  then
             $S := S'$ 
        end if
    end if
    if  $S$  is strictly better than  $S_{\text{best}}$  then
         $S_{\text{best}} := S$ 
    end if
    Compute  $T_{k+1}$ 
     $k := k + 1$ 
end while
 $S_{\text{best}}$  is the solution of the method

```

Figure 7.11. Principle algorithm of inhomogenous simulated annealing

Hajek proved, in [HAJ 88], that inhomogenous simulated annealing converges in probability towards the set of optimal solutions under certain hypotheses on the temperature and the neighboring system properties.

When the Markovian model can be used, $E(z(S))$ and $E(z(S'))$ are computed by using theorem 7.1:

$$\Delta z = E(z(S)) - E(z(S'))$$

otherwise, $E(z(S))$ and $E(z(S'))$ are estimated by using the algorithm described in Figure 7.8:

$$\Delta z = \overline{z(S)} - \overline{z(S')}$$

7.3.4. Kangaroo algorithm for the minimization in expectation

The kangaroo algorithm (Figure 7.12) was proposed in [FLE 93] with the aim of avoiding the tuning of the parameter of the simulated annealing and to avoid the convergence towards a local optimum with the stochastic descent. The algorithm allows us, after a stochastic descent without improvement for a given number of

```

Let  $A > 0$  be a maximum number of iterations without improvement
Let  $S$  be an initial solution
 $k := 0, S_{\text{best}} := S$ 
while necessary do
    if  $k < A$  then [Stochastic descent]
        Choose  $S'$  randomly and uniformly in the neighborhood  $\mathcal{V}$  of  $S$ .
        if  $S'$  is better than  $S$  then
             $k := 0$ 
            if  $S'$  is better than  $S_{\text{best}}$  then
                 $S_{\text{best}} := S'$ 
            end if
             $S := S'$ 
        else
             $k := k + 1$ 
        end if
    else [no improvement since  $A$  iterations]
        Choose  $S'$  randomly and uniformly in the neighborhood  $\mathcal{W}$  of  $S$ .
         $S := S'$ 
         $k := 0$ 
    end if
end while
 $S_{\text{best}}$  is the solution of the method

```

Figure 7.12. Kangaroo principle algorithm

iterations, to accept any solutions whatever the value of the objective function, and to restart a stochastic descent. Compared to the iterated stochastic descent, this algorithm does not lose the information about the local optimum found.

After a stochastic descent, with a neighboring system \mathcal{V} , if there is no improvement since A iterations, any neighbor in the neighborhood \mathcal{W} is accepted. The neighboring system \mathcal{W} is not necessarily the same as \mathcal{V} but must respect the accessibility property. The choice of a neighbor in the neighboring system \mathcal{W} is called a “jump”.

The neighboring system \mathcal{W} can be chosen in many ways. The easiest consists of $\mathcal{W} = \mathcal{V}$, a jump then corresponding to the acceptance of a “bad” solution. If $\mathcal{W}(S) = \Omega, \forall S \in \Omega$, the algorithm is the iterated stochastic descent. Another possibility consists of applying the neighboring system \mathcal{V} many times, which amounts to accepting, without condition, many successive transitions. The neighboring system \mathcal{W} can also be independently defined.

Contrary to the iterated local search, we obtain the following theoretical result:

THEOREM 7.3.- [FLE 93] *The kangaroo algorithm converges in probability towards the set of global minimums if and only if the neighboring system \mathcal{W} respects the accessibility property.*

The proof of the convergence lies in the fact that the kangaroo algorithm builds a Markov chain where any state can lead to an absorbing state and the absorbing states constitute the global optimal set.

7.3.5. Neighboring systems

Classical neighboring systems for the flow-shop problem are permutations and insertions:

$P_{j,j+1}$: permutation of two contiguous jobs T_j and T_{j+1} randomly chosen,

$P_{j,j'}$: permutation of any two jobs T_j and $T_{j'}$ randomly chosen,

$I_{j,j'}$: insertion of a job T_j randomly chosen at a new position j' randomly chosen.

They satisfy the accessibility and the reversibility properties. For the kangaroo algorithm, we use, for the neighboring system \mathcal{W} , five times the neighboring system \mathcal{V} . In the following, we use the neighboring system $I_{j,j'}$ which provides better results than the others.

7.4. Computational experiment

We have tested proposed methods (performance evaluation methods and scheduling methods) on flow-shop problems from the OR-Library (<http://mscmga.ms.ic.ac.uk/info.html>). OR-Library is a collection of test data sets for a variety of operations research problems. In this library, the data (job processing times p_{ij}) correspond to deterministic problems. Here, we have deduced from these data, processing times which follow exponential distribution, uniform distribution or normal distribution.

The number of replications of the simulation model has been empirically determined. For the three distribution functions, 5000 replications give good results in a reasonable computation time.

The four versions of the comparative process between two schedules have been tested. Detailed results are given in [GOU 03, GOU 01, GRA 01, GOU 05]. The best results have been obtained by the first version (used here) and the fourth version.

Unless otherwise instructed, the set of methods have been tested on flow-shop with unlimited buffers.

7.4.1. Exponential distribution

The rates of exponential distribution ($\mu_{i,j}$) are deduced from the processing times by:

$$\mu_{i,j} = 1/p_{i,j}$$

Table 7.4 presents a comparison of schedules obtained by the combination (stochastic descent – Markovian model) (1000 iterations) with optimal schedules obtained in a deterministic context. Each schedule is obtained by a stochastic descent without random variable: the descent is classically realized with processing times $p_{i,j}$ and stopped when an optimal schedule is obtained, this being evaluated with the Markovian model. For each instance, we give the number of jobs, the number of machines, the optimal makespan C_{\max} in a deterministic context, the mean of the expected makespan for ten runs $E(\bar{C}_{\max})$ and the corresponding standard deviation (SD).

Each of the ten runs of the stochastic descent without random variable provides an optimal schedule for the deterministic problem but the obtained schedules are not the same. The important standard deviation of the mean of the ten runs can be explained by the fact that two schedules with the same makespan in a deterministic context may have different expected makespans.

Instance	$(n \times m)$	Deterministic optimum			Stochastic descent Markovian model	
		C_{\max}	$E(\bar{C}_{\max})$	SD	$E(\bar{C}_{\max})$	SD
car1	(11 × 5)	7038	9658.81	80.48	9513.26	0.00
car2	(13 × 4)	7166	9878.54	50.92	9651.88	0.00
car3	(12 × 5)	7312	10593.33	46.08	10339.89	0.05
car4	(14 × 4)	8003	10861.71	73.82	10507.93	0.11
car5	(10 × 6)	7720	11316.32	23.57	10789.47	0.00
car6	(8 × 9)	8505	12708.70	0.00	12618.87	0.00
car7	(7 × 7)	6590	9524.13	0.00	9436.88	0.00
car8	(8 × 8)	8366	11988.24	0.00	12047.04	0.01
reC01	(20 × 5)	1247	1809.18	3.39	1781.71	0.01
reC03	(20 × 5)	1109	1572.13	6.30	1544.39	0.00
reC05	(20 × 5)	1245	1741.53	9.54	1691.6	0.01

Table 7.4. Combination (stochastic descent – Markovian model) and (deterministic optimum) for the problem $F_m | p_{i,j} \sim \exp(\mu_{i,j}) | E(C_{\max})$

Table 7.5 shows a comparison between the combination (stochastic descent – Markovian model) (10000 iterations) and the combination (stochastic descent – simulation model) (10000 iterations). The table gives, for each instance, the mean of the obtained expected makespan for ten runs of each combination and the mean computation time (CT) for obtaining the results (in seconds with an O2 Silicon graphics at 195 MHz).

Instance	$(n \times m)$	Stochastic descent			
		Markov		Simulation	
		$E(\bar{C}_{\max})$	CT	$E(\bar{C}_{\max})$	CT
car1	(11 × 5)	9513.26	825	9513.26	595
car2	(13 × 4)	9651.88	229	9653.31	562
car3	(12 × 5)	10339.89	1714	10339.89	698
car4	(14 × 4)	10507.93	557	10503.13	602
car5	(10 × 6)	10789.47	2920	10791.01	645
car6	(8 × 9)	12618.87	26282	12619.72	792
car7	(7 × 7)	9436.88	692	9438.19	520
car8	(8 × 8)	12047.04	8820	12048.71	679
reC01	(20 × 5)	1781.71	74955	1782.33	237
reC03	(20 × 5)	1544.39	234799	1545.39	237
reC05	(20 × 5)	1691.60	235140	1693.34	230

Table 7.5. Combinations (stochastic descent – Markovian model) and (stochastic descent – simulation model) for the problem Fm | $p_{i,j} \sim \exp(\mu_{i,j})$ | $E(C_{\max})$

Both combinations obtain similar results: the deviation between two means is less than 1%. Moreover, the standard deviations of the means are small for the two combinations. The computation times of the combination with the simulation model remains steady, whereas the computation times of the combination with the Markovian model have a large range. These depend only on the time to compute the expected makespan, which increases quickly with the number of machines and the number of jobs.

Table 7.6 presents a comparison between the combinations (stochastic descent – simulation model) (10000 iterations) and (kangaroo algorithm – simulation model) (10000 iterations). For each instance and each combination, the table gives the mean of the obtained results for the ten runs and the mean computation time to obtain the results (in seconds with an O2 Silicon graphics at 195 MHz). The size of the instances does not allow us to compute the expected makespan in a reasonable time with the Markovian model. The obtained schedules are compared using the simulation model with 100000 samples of processing times.

Instance	$(n \times m)$	Stochastic descent		Kangaroo algorithm	
		$E(\bar{C}_{\max})$	CT	$E(\bar{C}_{\max})$	CT
reC01	(20 × 5)	1782.34	236	1779.84	3090
reC03	(20 × 5)	1545.39	231	1545.39	3300
reC05	(20 × 5)	1693.34	231	1692.10	3802
reC07	(20 × 10)	2310.94	307	2310.63	6045
reC09	(20 × 10)	2299.04	307	2293.51	6117
reC11	(20 × 10)	2195.01	307	2190.14	6192
reC13	(20 × 15)	3000.18	442	2996.90	6044
reC15	(20 × 15)	2984.67	443	2978.82	4245
reC17	(20 × 15)	2984.62	443	2981.57	4231
reC19	(30 × 10)	3160.64	459	3153.10	4281
reC21	(30 × 10)	3022.87	462	3013.39	4318
reC23	(30 × 10)	3026.15	461	3013.68	4977
reC25	(30 × 15)	3932.09	663	3926.31	9294
reC27	(30 × 15)	3686.45	1251	3673.40	9300
reC29	(30 × 15)	3585.33	686	3570.46	9298
reC31	(50 × 10)	4462.75	764	4428.95	9798
reC33	(50 × 10)	4408.80	804	4375.48	9784
reC35	(50 × 10)	4477.04	763	4435.42	9788

Table 7.6. Combination (stochastic descent – simulation model) and (kangaroo algorithm – simulation model) for the problem Fm $|p_{i,j} \sim \exp(\mu_{i,j})|E(C_{\max})$

The two combinations give similar results. The combination (kangaroo algorithm – simulation model) provides, for each instance, better results than the combination (stochastic descent – simulation model), but the improvement remains quite small compared to the time required for obtaining it. For all instances, the standard deviation of the mean is small.

Table 7.7 shows a comparison between the combination (stochastic descent – Markovian model) (10000 iterations), (kangaroo algorithm – Markovian model) (10000 iterations) and (simulated annealing – Markovian model) for flow-shop problems without buffer and flow-shop with limited buffers. For each instance and for each method, we give the buffer size, the mean of the obtained results by ten runs. The three methods allow us to obtain similar results.

7.4.2. Uniform distribution function

We consider two ranges for the uniform distribution function: $[0.9 \times p_{i,j}; 1.1 \times p_{i,j}]$ and $[0.8 \times p_{i,j}; 1.2 \times p_{i,j}]$. These ranges correspond to a deviation of $+/- 10\%$ and $+/- 20\%$ of the processing times.

Instance	$(n \times m)$	$b_i, \forall i = 1, \dots, m$	Stochastic descent $\overline{E(C_{\max})}$	Simulated annealing $\overline{E(C_{\max})}$	Kangaroo algorithm $\overline{E(C_{\max})}$
car1	(11×5)	0	11615.70	11713.20	11615.70
		1	10770.39	10770.39	10770.39
		2	10386.57	10386.57	10386.57
car2	(13×4)	0	12328.69	12266.54	12328.69
		1	10513.59	10492.64	10492.64
		2	9962.60	9959.35	9962.60
car3	(12×5)	0	13210.87	13163.23	13163.23
		1	11494.46	11494.46	11494.46
		2	11049.07	11051.68	11049.07
car5	(10×6)	0	13745.36	13745.36	13745.36
		1	12009.34	11996.77	11996.76
		2	11578.52	11670.74	11578.52
car7	(7×7)	0	10212.60	10212.60	10212.6
		1	9517.03	9517.03	9517.03
		2	9410.21	9410.21	9410.21
reC01	(20×5)	0	2439.65	2426.46	2438.85
		1	2027.64	2023.45	2023.04
		2	1878.76	1948.10	1877.54

Table 7.7. Combinations (stochastic descent – Markovian model), (simulated annealing – Markovian model) and (kangaroo algorithm – Markovian model) for the problem $Fm | b_{i,i+1}, p_{i,j} \sim \exp(\mu_{i,j}) | E(C_{\max})$

Tables 7.8 and 7.9 present a comparison of the combination (stochastic descent – simulation model) (10000 iterations) with optimal schedules obtained in a deterministic context. The obtained schedules are compared by using the simulation model with 100000 samples of processing times. For each instance and each method, we give the optimal makespan in a deterministic context (C_{\max}), the mean of the estimation of the expected makespan for ten runs ($\overline{E(C_{\max})}$) and the corresponding standard deviation (SD).

When the range of the uniform distribution function is $+/-10\%$, the estimation of the expected makespan is close to the makespan in a deterministic context (from 0.02% to 1.27%). The impact of random events is weak on the makespan. Results obtained by the stochastic descent without taking into account random events are similar to the results obtained by the proposed combination. Taking into account the computation time, it is not worthwhile using such a combination.

Instance	$(n \times m)$	Deterministic optimum			Stochastic descent Simulation	
		C_{\max}	$E(\bar{C}_{\max})$	SD	$E(\bar{C}_{\max})$	SD
car1	(11 × 5)	7038	7074.13	22.09	7037.42	0.00
car2	(13 × 4)	7166	7208.81	6.90	7167.63	0.01
car3	(12 × 5)	7312	7357.73	17.67	7407.32	6.60
car4	(14 × 4)	8003	8048.74	21.49	8009.44	0.89
car5	(10 × 6)	7720	7791.55	0.02	7828.52	33.76
car6	(8 × 9)	8505	8613.09	0.00	8679.62	91.37
car7	(7 × 7)	6590	6663.60	0.00	6711.35	57.77
car8	(8 × 8)	8366	8446.60	0.00	8446.60	0.00
reC01	(20 × 5)	1247	1263.81	0.84	1271.41	6.24
reC03	(20 × 5)	1109	1122.42	1.37	1121.47	5.89
reC05	(20 × 5)	1245	1257.51	0.93	1266.06	8.23

Table 7.8. Combination (stochastic descent – simulation model) and (deterministic optimum) for the problem Fm | $p_{i,j} \sim U(0.9 \times p_{i,j}; 1.1 \times p_{i,j})$ | $E(C_{\max})$

Instance	$(n \times m)$	Deterministic optimum			Stochastic descent Simulation	
		C_{\max}	$E(\bar{C}_{\max})$	SD	$E(\bar{C}_{\max})$	SD
car1	(11 × 5)	7038	7158.56	38.27	7066.21	0.00
car2	(13 × 4)	7166	7306.42	14.47	7274.85	98.50
car3	(12 × 5)	7312	7481.94	20.89	7471.67	22.16
car4	(14 × 4)	8003	8147.89	31.94	8056.51	6.67
car5	(10 × 6)	7720	7937.79	0.25	8023.44	137.91
car6	(8 × 9)	8505	8775.34	0.00	8773.64	5.13
car7	(7 × 7)	6590	6762.21	0.00	6763.33	0.28
car8	(8 × 8)	8366	8589.76	0.00	8589.12	9.23
reC01	(20 × 5)	1247	1288.45	1.13	1303.28	27.92
reC03	(20 × 5)	1109	1142.28	2.35	1136.57	2.43
reC05	(20 × 5)	1245	1276.06	1.65	1278.26	4.29

Table 7.9. Combination (stochastic descent – simulation model) and (deterministic optimum) for the problem Fm | $p_{i,j} \sim U(0.8 \times p_{i,j}; 1.2 \times p_{i,j})$ | $E(C_{\max})$

When the range is $+/-20\%$, the deviation between the estimation of the expected makespan and the makespan in deterministic context increases (from 0.4% to 3.32%). The impact of random events is more important on the makespan. The proposed combination seems to obtain better results. Taking into account comments made concerning the exponential distribution function, we can suppose that the more the range increases, the better the results obtained by the combination.

7.4.3. Normal distribution function

We consider a normal distribution function with mean $p_{i,j}$ and standard deviation 20% $p_{i,j}$. As for the uniform distribution function, Table 7.10 presents a comparison of the combination (stochastic descent – simulation model) (10000 iterations) with optimum schedules obtained in a deterministic context.

Instance	$(n \times m)$	Deterministic optimum			Stochastic descent	
		C_{\max}	\overline{C}_{\max}	SD	\overline{C}_{\max}	SD
car1	(11 × 5)	7038	7329.01	55.61	7167.57	0.00
car2	(13 × 4)	7166	7482.88	29.08	7398.80	78.31
car3	(12 × 5)	7312	7709.59	24.09	7645.19	20.92
car4	(14 × 4)	8003	8335.06	43.56	8186.32	7.82
car5	(10 × 6)	7720	8188.24	0.54	8125.36	10.72
car6	(8 × 9)	8505	9037.29	0.00	9037.33	0.05
car7	(7 × 7)	6590	6942.35	0.00	6926.54	1.18
car8	(8 × 8)	8366	8833.82	0.00	8819.12	2.22
reC01	(20 × 5)	1247	1328.59	1.79	1327.41	15.11
reC03	(20 × 5)	1109	1174.38	3.27	1160.36	6.45
reC05	(20 × 5)	1245	1308.71	2.53	1301.55	1.68

Table 7.10. Combination (stochastic descent – simulation model) and (deterministic optimum) for the problem Fm | $p_{i,j} \sim \text{normal}(p_{i,j}; 0.2 \times p_{i,j}) | E(C_{\max})$

In this case, random events have more impact on processing times and so on the makespan. As supposed, the proposed combination, which takes random events into account, provides better results than the stochastic descent without taking random events into account.

7.5. Conclusion

The stochastic permutation flow-shop scheduling problem has been little studied. Moreover, as shown in state-of-the-art reviews, the problem has been studied, then abandoned, and studied again in the late 1990s. Nevertheless, many authors, among them Cunningham and Dutta [CUN 73], comment that in many practical situations, the time for a machine to realize a job is a random variable. Is this disaffection due to a lack of interest or to the underlying performance evaluation problem?

We have probably studied this problem because of the term stochastic which is one of our central themes of research: queuing network modeling and discrete event simulation. For us, this study is also a starting point for research into robustness problems [KOU 00], stochastic minimization problems and more general problems in an uncertain context (other distribution functions, other scheduling problems, supply chain, etc.).

7.6. Bibliography

- [BAG 70a] BAGGA P.C., “n jobs, 2 machines sequencing problems with stochastic service times”, *Operations Research*, vol. 7, p. 184–197, 1970.
- [BAG 70b] BAGGA P.C., “Sequencing with random service times”, *Technometrics*, vol. 12, p. 327–334, 1970.
- [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. 391–413, 1995.
- [CUN 73] CUNNINGHAM A.A. and DUTTA S.K., “Scheduling jobs with exponentially distributed processing times on two machines of a flow shop”, *Naval Research Logistics Quarterly*, vol. 16, p. 69–81, 1973.
- [FLE 93] FLEURY G., Méthodes stochastiques et déterministes pour les problèmes NP-difficiles, PhD Thesis, Blaise Pascal University, Clermont-Ferrand II, France, 1993.
- [FOL 84] FOLEY R.D. and SURESH S., “Stochastically minimizing the makespan in flow shops”, *Naval Research Logistics Quarterly*, vol. 31, p. 551–557, 1984.
- [FOR 81] FORST F.G., An analysis of the two machine static stochastic flow shop with linear completion time costs, PhD Thesis, University of Illinois at Urbana Campaign, 1981.
- [FOR 83] FORST F.G., “Minimizing total expected costs in the two machine stochastic flow shop”, *Operations Research Letters*, vol. 2, p. 58–61, 1983.
- [FOR 84] FORST F.G., “A review of the static stochastic job sequencing literature”, *Operations Research*, vol. 21, p. 127–144, 1984.
- [GOU 00] GOURGAND M., GRANGEON N. and NORRE S., “A review of the static stochastic flow shop scheduling problem”, *Journal of Decision Systems*, vol. 9, no. 2, p. 183–214, 2000.
- [GOU 01] GOURGAND M., GRANGEON N. and NORRE S., “Markovian analysis for performance evaluation in two machine stochastic flow shop”, *International Conference on Industrial Engineering and Production Management (IEPM 01)*, Quebec, 20–23 August, 2001.

- [GOU 03] GOURGAND M., GRANGEON N. and NORRE S., “A contribution to the stochastic flow shop scheduling problem”, *European Journal of Operational Research*, vol. 151, no. 2, p. 415–433, December 2003.
- [GOU 05] GOURGAND M., GRANGEON N. and NORRE S., “Markovian analysis for performance evaluation and scheduling in M machine stochastic flow shop with buffers of any capacity”, *European Journal of Operational Research*, vol. 161, no. 1, p. 126–147, 2005.
- [GRA 01] GRANGEON N., Métaheuristiques et Modèles d’Evaluation pour le Problème du Flow-Shop Hybride Hiérarchisé, PhD Thesis, Blaise Pascal University, Clermont-Ferrand II, France, 2001.
- [HAJ 88] HAJEK B., “Cooling schedules for optimal annealing”, *Mathematics of Operations Research*, vol. 13, p. 311–329, 1988.
- [HAO 99] HAO J.K., GALINIER P. and HABIB M., “Métaheuristiques pour l’optimisation combinatoire et l’affectation sous contraintes”, *Revue d’Intelligence Artificielle*, vol. 13, no. 2, p. 283–324, 1999.
- [JIA 98] JIA C., “Minimizing variation in stochastic flow shop”, *Operations Research Letters*, vol. 23, p. 109–111, 1998.
- [KAL 06] KALCZYNSKI P.J. and KAMBUROWSKI J., “A heuristic for minimizing the expected makespan in two-machine flow shops with consistent coefficients of variation”, *European Journal of Operational Research*, vol. 169, p. 742–750, 2006.
- [KAM 99] KAMBUROWSKI J., “Stochastically minimizing the makespan in two-machine flow shops without blocking”, *European Journal of Operational Research*, vol. 112, p. 304–309, 1999.
- [KIR 83] KIRKPATRICK S., GELATT C.D. and VECCHI M.P., “Optimization by simulated annealing”, *Science*, vol. 20, p. 671–680, 1983.
- [KOU 00] KOUVELIS P., DANIELS R. and VAIRAKTARAKIS G., “Robust scheduling of a two machine flow shop with uncertain processing times”, *IIE Transactions*, vol. 32, p. 421–432, 2000.
- [KU 86] KU P.S. and NIU S.C., “On Johnson’s two machine flow shop with random processing times”, *Operations Research*, vol. 34, p. 130–136, 1986.
- [MAK 65] MAKINO T., “On a scheduling problem”, *Journal of the Operations Research Society of Japan*, vol. 8, p. 32–44, 1965.
- [MIT 77] MITTAL B.S. and BAGGA P.C., “A priority problem in sequencing with stochastic services times”, *Operations Research*, vol. 14, p. 19–28, 1977.
- [PIN 82] PINEDO M., “Minimizing the expected makespan in stochastic flow shops”, *Operations Research*, vol. 30, p. 148–162, 1982.

- [PIN 95] PINEDO M., *Scheduling: Theory, Algorithms and Systems*, Prentice Hall, Englewood Cliffs, NJ, 1995.
- [POR 06] PORTOUGAL V. and TRIETSCH D., “Johnson’s problem with stochastic processing times and optimal service level”, *European Journal of Operational Research*, vol. 169, p. 751–760, 2006.
- [PRA 81] PRASAD V.R., “nx2 flow shop sequencing problem with random processing times”, *Operations Research*, vol. 18, p. 1–14, 1981.
- [QNA 94] QNAP2, *Manuel de référence*, Copyright BULL and INRIA, 1994.
- [SHA 94] SHAKED M. and SHANTHIKUMAR J.G., *Stochastic Orders and their Applications*, Academic Press, Boston, 1994.
- [TAL 67] TALWAR T.T., “A note on sequencing problems with uncertain job times”, *Journal of the Operations Research Society of Japan*, vol. 9, 1967.
- [WEI 82] WEISS G., “Multiserver stochastic scheduling”, in *Deterministic and Stochastic Scheduling*, DEMPSTER M.A.H., LENSTRA J.K. and RINNOOY KAN A.H.G. (Eds.), D. Reidel, Dordrecht, p. 157–179, 1982.

Chapter 8

Resource Allocation for the Construction of Robust Project Schedules

This chapter deals with disruption management in resource-constrained project scheduling. We consider two types of uncertainties: increased activity duration and insertion of an unexpected activity. A reactive policy based on resource allocation decisions is defined to repair a static precomputed schedule. These decisions are represented through a network flow model. To tackle the first type of uncertainty, we propose a proactive resource allocation method aiming at maximizing the baseline schedule stability. The stability criterion is defined as the weighted sum of the expected start time deviations between the actually realized schedule and the predictive schedule. Optimal flows are computed by means of a branch-and-bound method, integrating constraint propagation techniques. To deal with the second type of uncertainty, we propose a predictive/reactive method. A polynomial algorithm is proposed to insert the unexpected activity in the precomputed resource flow network while minimizing the makespan increase. The efficiency of the proposed methods is shown using computational experiments.

8.1. Introduction

The resource-constrained project scheduling problem (RCPSP), denoted $PS \mid prec \mid C_{max}$ according to the notation of Chapter 1, has received increasing attention during the last decades. For state-of-the-art surveys, we refer to

Chapter written by Christian ARTIGUES, Roel LEUS and Willy HERROELEN.

[BRU 99, HER 98, KOL 01, ART 08b]. The vast majority of research effort concentrates on exact and heuristic solution methods in a static and deterministic environment. In this context, start times are assigned to activities, generally with the objective of minimizing the makespan. The resulting schedule, called in this chapter the *baseline schedule*, is used to guide the project implementation in a dynamic environment.

However, during project execution, activities are subject to uncertainty which may greatly disrupt the initial solution. Such uncertainty comes from many different possible sources: activities may last longer than expected, resources may become unavailable, raw materials may be delivered late, workers may be absent, new activities may have to be inserted, etc. Despite the variety of the possible sources, we assume here that uncertainty can be represented by stochastic activity processing times and by the possibility of activity occurrence after the baseline schedule is issued. These two types of uncertainty cover multiple actual disruptions.

Mehta and Uzsoy [MEH 98] underline that the baseline schedule has two major functions. The first one is resource allocation to the different activities so as to optimize the workshop performance. The second one, which is also cited in [WU 93], is to serve as a basis for external activity planning, such as material supply, preventive maintenance, and due date announcement to customers. In multi-project environments, establishing a baseline schedule approved in advance by all parties (clients, suppliers, employees and other resources) is crucial. Subcontractors can for example commit to perform the required work inside a pre-established time window. The baseline schedule is of primary importance for budget projection and for employee performance evaluation after project completion. A more complete study of the baseline schedule functions can be found in [AYT 05]. Consequently, the baseline schedule is in charge of guiding all human resources involved during project execution. Therefore, its stability is a necessity. For more details on stability in scheduling we refer to Chapter 1 and to [HER 04, LEU 03]. For robust or proactive/reactive project scheduling, we refer to [HER 05, DEM 08].

For the sake of simplicity, we consider only one resource type, which can be selected as the most restrictive resource, constituting the organization bottleneck. The chapter is organized as follows. Resource allocation solutions, corresponding to transportation network flows, are presented in section 8.2. Section 8.3 describes a branch-and-bound method issuing a robust resource allocation. The algorithm exploits constraint propagation techniques as well as an efficient procedure for verifying the existence of a feasible flow. Section 8.4 is dedicated to an optimal

reactive method in response to an unexpected activity arrival, while respecting the initial resource allocation. In both cases, the proposed methods are validated using computational results. Section 8.5 provides concluding remarks and suggestions for future work. Note that a different approach from the dynamic RCPSP is presented in Chapter 14. While resource allocation is not explicitly represented, a predictive/reactive method based on constraint programming with explanations allows for a re-optimization of a solution in the presence of various disruptions.

8.2. Resource allocation and resource flows

We present some mathematical notations used throughout this chapter (section 8.2.1). The representation of resource allocation by means of a flow in a transshipment network is the subject of section 8.2.2 and a simple method for the generation of a feasible flow starting from a given schedule is described in section 8.2.3. The way in which we propose to react to deviations in the input data is explained in section 8.2.4.

8.2.1. Definitions and notation

It is assumed that a set of activities N is to be scheduled on a single renewable resource type with availability a . The activities are numbered from 0 to n ($|N| = n + 1$) and activity i has fixed baseline duration $p_i \in \mathbb{N}$ and requires $R_i \in \mathbb{N}$ units of the single resource type, with $R_i \leq a$. Apart from the dummy start activity 0 and dummy end n , activities have a non-zero duration; the dummies also have zero resource usage. A is the set of pairs of activities between which a finish-start precedence relationship with time lag 0 exists. We assume graph $G(N, A)$ to be acyclic and equal to its transitive reduction (no redundant arcs are included). Without loss of generality, we also require $\forall(i, j) \in A, i < j$. For any $X \subset N \times N$, we can obtain the immediate predecessors of activity i by function $\pi_X : N \mapsto 2^N : i \mapsto \pi_X(i) = \{j \in N \mid (j, i) \in X\}$, and its direct successors by $\sigma_X : N \mapsto 2^N : i \mapsto \sigma_X(i) = \{j \in N \mid (i, j) \in X\}$, and we define TX to be the transitive closure of X , meaning that $(i, j) \in TX$ if a path from i to j exists in $G(N, X)$. Following Ford and Fulkerson [FOR 62] and to simplify the notation, we adopt the following conventions. For $X, Y \subseteq N$, let (X, Y) denote the set of all arcs that lead from $x \in X$ to $y \in Y$ in the graph under consideration, and for any function g on $N \times N$, let $g(X, Y) = \sum_{(x,y) \in (X,Y)} g(x, y)$. Similarly, when dealing with a function h defined on the elements of N , we put $h(X) = \sum_{x \in X} h(x)$. We may denote a set consisting of one element by its single element and omit duplicated brackets.

A schedule \mathcal{S} is defined by an $(n+1)$ vector of start times $s = (s_0, \dots, s_n)$; every s implies an $(n+1)$ vector of finish times C , $C_i = s_i + p_i, \forall i \in N$. Alternatively we write $s(\mathcal{S})$ and $C(\mathcal{S})$ for indicating the schedule. With every schedule \mathcal{S} , we associate a set $\delta(\mathcal{S})$ of time instants that correspond with the activity finish times: $t \in \delta(\mathcal{S})$ if and only if $\exists i \in N, t = C_i$. Define $N_t = \{i \in N \mid s_i < t \leq C_i\}$, the activities that are active during period t . The schedule \mathcal{S} is feasible if

$$(1) \forall (i, j) \in A, C_i(\mathcal{S}) \leq s_j(\mathcal{S}), \quad \text{and} \quad (2) \forall t \in \delta(\mathcal{S}), R(N_t) \leq a \quad (8.1)$$

An RCPSP-instance (N, A, a, p, R) aims to find a feasible schedule that minimizes C_n (in this case for a single resource type), where vector p collects the activity durations and R the resource requirements.

8.2.2. Resource flow networks

Artigues and Roubellat [ART 00] present *resource flow networks*, in which the amount of resources being transferred immediately from one activity to another is explicitly recorded. A similar representation was used in [BOW 95, NAE 89, NEU 02, FOR 97]. Let $u_i = R_i, \forall i \in N \setminus \{0, n\}$, and $u_0 = u_n = a$. A resource flow f associates with each pair $(i, j) \in N \times N$ a value $f_{ij} = f(i, j) \in \mathbb{N}$. These values must satisfy the following constraints, which constitute flow conservation and lower and upper bounds on node flow:

$$f(i, N) = u_i \quad \forall i \in N \setminus \{n\} \quad (8.2)$$

$$f(N, i) = u_i \quad \forall i \in N \setminus \{0\} \quad (8.3)$$

f_{ij} represents the (discrete) number of resources that are transferred from activity i to activity j . For a flow f , we define the set of activity pairs $\Phi(f) = \{(i, j) \in N \times N \mid f_{ij} > 0\}$, containing the arcs that carry flow in the resource flow network. We also define $\chi(f) = \Phi(f) \setminus TA$: the arcs in $\chi(f)$ are the flow-carrying arcs that do not represent technological precedence relations. In other words, $\chi(f)$ are extra “resource arcs” that induce extra precedence constraints. For any $X \subset N \times N$, we let G_X represent the graph $G(N, TA \cup X)$. We call flow f feasible when condition (8.4) holds: the additional precedence constraints implied by $\chi(f)$ do not prevent execution of the project if f is feasible.

$$G_{\chi(f)} \text{ is acyclic} \quad (8.4)$$

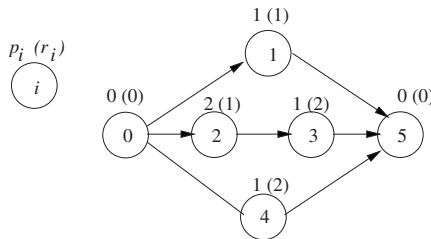


Figure 8.1. Example project network

EXAMPLE.— A small example allows us to illustrate the definitions given above. In Figure 8.1, an example network is represented in activity-on-the-node format, and we assume $a = 3$. According to our definitions, $u_0 = u_5 = 3$, $u_1 = u_2 = 1$ and $u_3 = u_4 = 2$. One possible resource flow is illustrated in Figure 8.2(a). We see for instance that one of the available resource units is transferred from the end of dummy activity 0 to the start of activity 2. This unit is released at the completion of activity 2 and transferred to the start of activity 3. The resource flow network shown in Figure 8.2(b) represents an alternative resource allocation. For Figure 8.2(a), $\chi(f) = \{(4, 1), (4, 3)\}$ while $\chi(f) = \{(1, 3), (4, 1)\}$ in the resource flow network of Figure 8.2(b). The arcs in $\chi(f)$ are dashed.

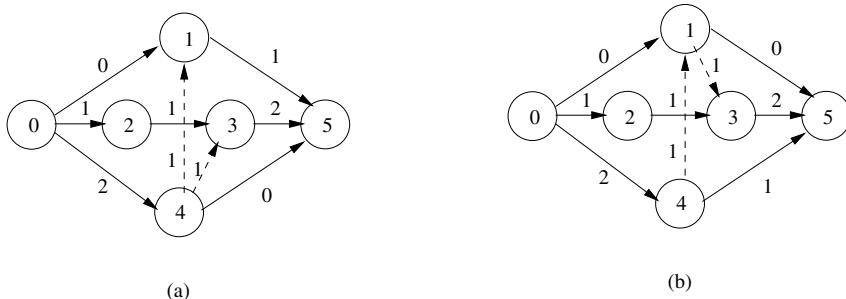


Figure 8.2. Two resource flow networks. Flow values are indicated on the arcs

Define the earliest start schedule $\Theta(X)$, $X \subseteq N \times N$, to be the schedule in which $s_0 = 0$ and each other activity i starts at time $s_i = \max_{j \in \pi_{A \cup X}(i)} (s_j + p_j)$, provided that graph $G(N, A \cup X)$ is acyclic: the arcs in X represent extra precedence constraints, in addition to A . A solution to an RCPSP-instance can be obtained by finding a feasible flow f that minimizes $s_n(\Theta(A \cup \chi(f)))$, and we see that we obtain an extension of the disjunctive graph representation of the classical job-shop scheduling problem [ROY 64]. Both of the resource flows described in Figure 8.2 result in the same schedule depicted in Figure 8.3.

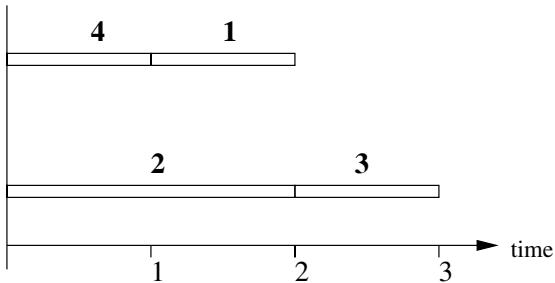


Figure 8.3. A schedule for the example of Figure 8.1

8.2.3. A greedy method for obtaining a feasible flow

Artigues *et al.* [ART 03] present a simple method for obtaining a feasible resource flow by extending a parallel schedule generation scheme (see for example [KOL 98]) to derive the flows during scheduling. This allocation routine is easily uncoupled from the schedule generation, and can be used to obtain a feasible flow for a given schedule \mathcal{S} . The procedure is described here for illustrative purposes, but will also be used for comparison with the branch-and-bound algorithm presented in section 8.3, and as a generation method for an initial flow before arrival of an unexpected activity (section 8.4).

The greedy algorithm uses a list \mathcal{L} of the activities, arranged in non-decreasing order of the starting times in schedule \mathcal{S} . At the start, all flows are initialized at 0, except the flow between 0 and n , which is initialized at $u_0 = u_n = a$. The flow is stepwise generated by inserting the activities in the flow following the order dictated by the list \mathcal{L} . More precisely, the computations for each task j are the following. The already included activities (including 0) are scanned in lexicographic order and for each activity k that is finished at or before the start of j , a value q is removed from f_{kn} , with $q = \min(u_j - f(N, j), f_{kn})$, and then $f_{kj} = f_{kj} + q$ and $f_{kn} = f_{kn} - q$. The insertion of j ends when $f(N, j) = u_j$. Then let $f_{jn} = u_j$. This algorithm has time complexity $\mathcal{O}(n^2)$.

EXAMPLE.– Consider the schedule given in Figure 8.3 and list $\mathcal{L} = (2, 4, 1, 3)$. The algorithm generates the flow represented in Figure 8.2(b).

8.2.4. Reactions to disruptions

The stochastic variable P_i represents the duration of activity $i \in N$. These variables are collected in vector P . When information becomes known about

durations P_i that take on a realization different from p_i (the baseline duration), the schedule needs to be repaired. The same holds when a new unexpected activity arrives and needs to be inserted into the baseline schedule. In this schedule repair process, we lay down that the resource allocation remains constant, i.e. the same resource flow is maintained. This reactive scheduling policy is preferred when specialized resources (e.g. expert staff) cannot easily be transferred between activities at short notice, for instance in a multi-project environment, where it is necessary to book key staff or equipment in advance to guarantee their availability [BOW 95]. This makes last-minute changes in resource allocation impossible, contrary to the case of totally dedicated resources. Artigues and Roubellat [ART 00] also refer to the desire to ensure schedule stability (avoiding system nervousness resulting from complete rescheduling – this objective will be more extensively discussed in section 8.3), and limited computation time, especially in the case of on-line scheduling.

EXAMPLE.— As an illustration, we again use the resource flow networks in Figure 8.2. Suppose that project management is uncertain about the duration of activity 4. It is obvious that in this case, the flow pattern in Figure 8.2(a) is more robust with regard to the expected makespan (currently equal to 3) than the pattern in Figure 8.2(b), which was generated by the greedy algorithm presented in section 8.2.3: with respect to a projected makespan of 3 time units, activity 4 has a total slack of 1 time unit in the first resource allocation, and 0 in the second. As a result, an increase in p_4 will have an immediate impact on the makespan of the repaired schedule in Figure 8.2(b), while a buffer of size 1 is provided in Figure 8.2(a).

Igelmund and Radermacher [IGE 83b] present different scheduling policies for stochastic project networks under resource constraints, all based on the concept of forbidden sets, which are sets of activities unrelated to procedure that are not allowed to be scheduled simultaneously because of resource constraints. In this chapter, we are interested in earliest start policies (ES-policies). The idea is to extend the given partially ordered set $G(N, A)$ to a partially ordered set $G(N, A \cup X)$ such that no forbidden set remains precedence unrelated and can thereby be scheduled in parallel. The condition for feasibility of the policy is that $G(N, A \cup X)$ still be acyclic. Then, in order to obtain a feasible schedule $\mathcal{S}(p)$ for a given scenario p of activity durations, an ES-policy simply computes earliest activity start times in the graph by performing a forward CPM (longest path) pass [STO 01]. For a new activity, we proceed similarly, requiring that the activity be appropriately inserted into the flow network.

The following theorem is intuitive and establishes a relation between flow networks and ES-policies. A proof of theorem 8.1 is given in [LEU 04], based in part on [MÖH 85].

THEOREM 8.1.– *For any feasible resource flow f , $X = \chi(f)$ defines a feasible ES-policy. Conversely, if X defines a feasible ES-policy, then a feasible flow f exists with $TA \cup \chi(f) \subseteq T(A \cup X)$.*

8.3. A branch-and-bound procedure for resource allocation

This section describes an algorithm for resource allocation with an eye on protection against variability in the activity durations for a given baseline schedule. Section 8.3.1 deals with the ties between duration perturbations and the concept of stability. Section 8.3.2 states the problem to be solved and explains the branching procedure. Further details of the search procedure are provided in section 8.3.3. Section 8.3.4 presents a test for the existence of a feasible flow in a given network. The branching rule is further outlined in section 8.3.5. Finally, section 8.3.6 presents computational results.

8.3.1. Activity duration disruptions and stability

We lay down the constraint that a resource allocation decision is compatible with an input schedule (the baseline schedule) \mathcal{S} : this compatibility guarantees that the baseline schedule will effectively be realized if everything goes as planned. Define $\Lambda(\mathcal{S}) = \{(i, j) \in N \times N | (i, j) \notin TA, i \neq j, C_i(\mathcal{S}) \leq s_j(\mathcal{S})\}$. A feasible flow f is said to be compatible with a feasible schedule \mathcal{S} , written $f \sim \mathcal{S}$, if $\forall (i, j) \in TA \cup \chi(f), C_i(\mathcal{S}) \leq s_j(\mathcal{S})$, or in other words if $\chi(f) \subseteq \Lambda(\mathcal{S})$. The pre-schedule \mathcal{S} is built before the resource allocation.

In situations where a pre-schedule is valuable it will often be of interest that activities are not started as soon as is feasible. Rather, we should attempt to respect the pre-schedule as far as possible in order to avoid system nervousness and constant resource rescheduling, in other words, to maintain the stability in the system. As a result, activities are started at the maximum of the ending times of the predecessors and their pre-schedule starting time. Other scheduling disciplines that operate in this way are railway and airline scheduling, crew rostering, etc. The actual starting time of activity i is a stochastic variable

$$S_i(P, \chi(f), \mathcal{S}) = \max \left(s_i(\mathcal{S}), \max_{j \in \pi_{TA \cup \chi(f)}(i)} (S_j(P, \chi(f), \mathcal{S}) + P_j) \right),$$

with $s_0(\mathcal{S}) = 0$. Following [HER 04], we adopt as a measure of pre-schedule stability the expected weighted deviation in start times in the actual schedule from those in

the pre-schedule (see Chapter 1). Our aim is to construct a feasible flow f with $\chi(f) \subseteq \Lambda(\mathcal{S})$ such that

$$E \left[\sum_{i \in N} c_i \times (S_i(P, \chi(f), \mathcal{S}) - s_i(\mathcal{S})) \right] = g(\chi(f)) \quad (8.5)$$

is minimized, where $E[\cdot]$ is the expectation operator and schedule \mathcal{S} is an input parameter. $c_i \in \mathbb{N}$ denotes the non-negative cost per unit time overrun on the start time of activity i , which reflects either the difficulty in obtaining the required resources (internal stability) or the importance of the activity delivering on time to the customer (external stability). We always set $c_0 = 0$; minimization of the expected makespan is the special case $c_i = 0, i < n$ and $c_n > 0$.

8.3.2. Problem statement and branching scheme

In section 8.3, the set of decision variables is the set of flows f_{ij} with $(i, j) \in F = TA \cup \Lambda(\mathcal{S})$. For any $(i, j) \in F$, B_{ij} is the domain within which f_{ij} can assume values. The objective is to minimize expression (8.5) subject to constraint sets (8.2) and (8.3) and the requirement that $f \sim \mathcal{S}$, which is implicit from the choice of F . Condition (8.4) is also satisfied because each arc $(i, j) \in F$ has $C_i(\mathcal{S}) \leq s_j(\mathcal{S}) \leq C_j(\mathcal{S})$, since input schedule \mathcal{S} is feasible. For $f_{ij} \in F$, B_{ij} can be represented by its lowest entry LB_{ij} and highest entry UB_{ij} : we represent the domains as intervals. We propose a branch-and-bound algorithm to implicitly search all valid flow values. The search procedure relies on constraint propagation for search space reduction.

We find an optimal resource allocation for a schedule \mathcal{S} by considering all subsets $\Omega \subseteq \Lambda(\mathcal{S})$ that allow a feasible flow in network $TA \cup \Omega$. Each such set corresponds with at least one and mostly multiple feasible f , with $\chi(f) \subseteq \Omega$. We iteratively add arcs from $\Lambda(\mathcal{S})$ to Ω until a feasible flow is attainable (the feasibility test is the subject of section 8.3.4). [LEU 03] and [LEU 04] show that the intuitive restriction to subset minimal sets Ω (when Ω admits a feasible flow, then other sets containing Ω need no longer be examined) and the restriction to the integer numbers contained in the interval domains of the flows does not remove all optimal solutions.

In each node k of the search tree, the set $F = TA \cup \Lambda(\mathcal{S})$ is partitioned into three disjoint subsets: $F = \alpha_k \cup \nu_k \cup \omega_k$, with $\alpha_k = \{(i, j) \in F, LB_{ij} > 0\}$ the set of included arcs, $\nu_k = \{(i, j) \in F, UB_{ij} = 0\}$ the set of forbidden arcs, and $\omega_k = \{(i, j) \in F, LB_{ij} = 0 \text{ and } UB_{ij} > 0\}$ the set of undecided arcs. The

bounds LB_{ij} and UB_{ij} are established using constraint propagation in conjunction with branching decisions. We add all arcs in $\alpha_k \setminus TA$ to Ω_k , which results in *partial network* $G_k = G_{\Omega_k}$. If a feasible flow can be obtained in G_k , we find the current search node and backtrack, otherwise we need further branching decisions. The branching decision itself entails the selection of an undecided arc $(i, j) \in \Lambda(\mathcal{S}) \cap \omega_k$: the left branch is to set $LB_{ij} = 1$, so as to include (i, j) in the partial network G_k . The right branch is to impose $UB_{ij} = 0$, so as to forbid any flow across (i, j) and prohibit inclusion of (i, j) in Ω by placing the arc into set ν_k . Indeed, by adding a new constraint, we split up the domain into two disjoint subsets, one of which is singleton $\{0\}$. We elaborate on the selection of the branching arc in section 8.3.5.

Branch-and-bound algorithms have been proposed for various classes of policies for the stochastic RCPSP [IGE 83a, STO 01]. For ES-policies, these are based on the forbidden set branching scheme, which proceeds as follows. The minimal forbidden sets (MFSS) are arranged in a pre-determined order L . Each node v in the search tree is associated with an MFS m and branching on v systematically resolves m by creating a child node w_{ij} of v for each ordered pair (i, j) , $i, j \in m$, $i \neq j$. Each leaf v of the search tree represents a policy that is defined by resolving each MFS according to the decisions made on the path from v to the tree root.

EXAMPLE.— The project of Figure 8.1 has 2 MFSS, namely $\{1, 2, 4\}$ and $\{3, 4\}$, which immediately gives us arcs $(4, 1)$ and $(4, 3)$ as only possible solution when \mathcal{S} is \mathcal{S}^* – the schedule given in Figure 8.3, which is a strong argument in favor of this strategy.

We note nevertheless that the extension of the so-called binary branching as proposed above with constraint propagation also excludes the need for branching in the example; for an illustration and theoretical comparisons between the two branching strategies, we again refer to [LEU 04].

8.3.3. Details of the branch-and-bound algorithm

By Jensen's inequality, the deterministic value obtained when activity durations are set equal to their expected values is a lower bound for our objective function (see [FUL 62] for an application to expected makespan bounding). In order to obtain a lower bound at every node of the search tree, we maintain a set of earliest starting times in G_k based on expected activity durations; these earliest starting times are continuously updated. We refer to this bound as the *critical path lower bound*.

Combinations of the precedence relations defined by $TA \cup \Omega_k$ imply extra transitive relations, captured by $T(A \cup \Omega_k)$. These implicit precedences are incurred anyway, so we can extend set $\Omega_k = T(A \cup \Omega_k) \setminus \nu_k$ without deteriorating the objective.

Stork [STO 01] presents a single machine relaxation bound for stochastic project scheduling. This bound considers sets of precedence unrelated activities that are pair-wise incompatible because of resource constraints and computes a lower bound for expected project makespan as the sum of the expected durations of the activities, plus some additional terms. In our problem, the sequencing problem for such sets of activities have already been completely solved: either directly or transitively, a precedence constraint $i \rightarrow j$ will be included for all pairs of incompatible activities (i, j) with $C_i(\mathcal{S}) \leq s_j(\mathcal{S})$. We can include all those pairs into Ω_0 from the outset (in our implementation, we add them to A). We refer to this pre-processing measure as the *single machine rule*.

EXAMPLE.– For the project in Figure 8.1, we see that activities 3 and 4 jointly consume more than the available three resource units. The schedule in Figure 8.3, referred to as \mathcal{S}^* , solves this conflict by positioning activity 4 before activity 3. We can therefore add element (4,3) to Ω_0 or A .

When too many arcs have been forbidden, the partial network can no longer be completed to generate a feasible flow. Fast detection of these situations allows us to stop exploring the current branch of the search tree. For this purpose, we resort to a second network: the *remainder network* $G_k^\nu = G_{\Lambda(\mathcal{S}) \setminus \nu_k}$. As long as G_k^ν allows a feasible flow respecting the branching decisions higher in the search tree, it is possible to select a set $\chi(f) \subseteq \Lambda(\mathcal{S}) \setminus \nu_k$ that allows a feasible flow in G_k and corresponds with all branching decisions. In this case, we apply constraint propagation to further tighten the domains of the decision variables and to avoid branching into unfeasible areas as well as making branching decisions that are already implicit.

The evaluation of the objective function $g(\chi(f))$ for a given flow f amounts to the PERT problem, which cannot be efficiently solved [HAG 88, MÖH 00]. For this reason, we usually approximate the expectation of the objective function of a given policy by means of simulation [IGE 83a, MÖH 89, STO 01]. For this reason, the branch-and-bound algorithm can be considered to be an approximation algorithm. In our algorithm, when we obtain a feasible set Ω_k (by extension, we call Ω_k feasible when the corresponding flow is feasible), we do not compute $g(\chi(f))$ for a feasible flow f in $TA \cup \Omega_k$, but rather $g(\Omega_k)$; logically $g(\chi(f)) \leq g(\Omega_k)$. We can show that this approach does not change the results of the algorithm.

8.3.4. Testing for the existence of a feasible flow

In this section we discuss a simple way to test for the existence of a feasible flow in a given network using maximal flow computations in a transformed network. Möhring [MÖH 85] studies a related transformation that, in our terminology, allows us to determine the minimal required value of a . Naegler and Schoenherr [NAE 89] and Neumann *et al.* [NEU 02] discuss similar transformations.

For network G_Ω , $\Omega \subseteq \Lambda(\mathcal{S})$, we construct a new network G'_Ω as follows. We switch from bounds on node flow (quantities u_i) to bounds on arc flow by duplicating each node $i \in N \setminus \{0, n\}$ into two nodes i_s and i_t and adding arc (i_t, i_s) to the network, with upper bound on the flow $f(i_t, i_s)$ equal to its lower bound, both equal to u_i (see [FOR 62]); nodes 0 and n are renamed 0_s and n_t , respectively. All arcs entering i in G_Ω now lead to i_t in G'_Ω ; all arcs leaving i now emanate from i_s .

Node i_t can be interpreted as the start of activity i (reception of resources), node i_s as its completion (passing on the resources to successor activities). We augment network G'_Ω with source node s , sink node t and arc (t, s) . We construct a new network G''_Ω by replacing each arc (i_t, i_s) in G'_Ω by the arcs (i_t, t) and (s, i_s) . The capacity function c assumes the following values: $c(s, i_s) = c(i_t, t) = u_i$, $\forall i \in N$, all other capacities are equal to $+\infty$. All flow lower bounds are set to 0.

EXAMPLE.– Figure 8.4(a) shows the transformed network G'_0 of G_0 for the project of Figure 8.1. We choose $\mathcal{S} = \mathcal{S}^*$ and $\Omega_0 = \{(4, 3)\}$ as suggested in section 8.3.2. Figure 8.4(b) shows the network G''_0 obtained from network G'_0 of Figure 8.4(a).

Denote by $\mu(\Omega)$ the maximal s - t flow value in G''_Ω and let h be a corresponding maximal flow. It is clear that h satisfies the following two conditions:

$$h(i_s, \{j_t \mid j \in N\}) \leq u_i \quad \forall i \in N \setminus \{n\} \quad (8.6)$$

$$h(\{j_s \mid j \in N\}, i_t) \leq u_i \quad \forall i \in N \setminus \{0\} \quad (8.7)$$

If we define $\mu_{\max} = a + R(N)$, we see that $\mu(\Omega) \leq \mu_{\max}$, and equality $\mu(\Omega) = \mu_{\max}$ holds if and only if a maximal s - t flow in G''_Ω saturates all source and sink arcs, so that conditions (8.6) and (8.7) are satisfied as equality. It can be shown that the following lemma holds (for a formal proof, see [LEU 04]):

LEMMA 8.1.– For $\Omega \subseteq \Lambda(\mathcal{S})$, a feasible flow f exists in G_Ω with $\chi(f) \subseteq \Omega$ if and only if $\mu(\Omega) = \mu_{\max}$.

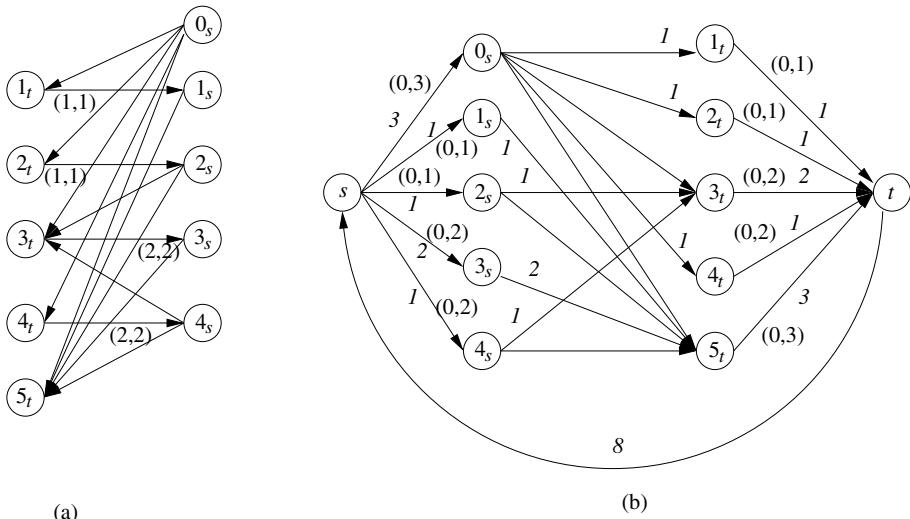


Figure 8.4. G'_0 and G''_0 for the example. Lower and upper bounds on arc flow are as indicated between brackets, otherwise $(0, +\infty)$. Flow values in (b) are indicated in italic, otherwise 0

The maximal flow in network G''_0 of Figure 8.4(b) amounts to $8 < \mu_{\max} = 9$ (flow quantities are indicated in the figure), so we conclude that a feasible flow is not attainable in G_0 . We apply lemma 8.1 during the course of the branch-and-bound algorithm to test for the existence of a feasible flow in both the partial network G_k and the remainder network G''_k . For this purpose, we use the extended networks G''_k and G'''_k .

At level 0 of the search tree, $f(i_s, j_t)$ in both networks is initialized at LB_{ij} for every (i, j) present, and we use a simple and efficient version of the traditional labeling algorithm to maximize flow, the shortest augmenting path algorithm [EDM 72, AHU 93]. This algorithm is strongly polynomial and is implemented in a breadth first approach, such that the labels are immediately available for use in the branching rule (see section 8.3.5). The flows in the two extended networks are maintained on an incremental basis rather than recalculated restarting from the lower bounds every time a feasibility check is required.

8.3.5. Branching rules

In this section we present a binary branching scheme that uses information about the maximum flow computations in the partial and remainder network. In order to obtain an increase in flow in G''_k , the branching arc itself or one of the other arcs

that are added to Ω_k needs to create a new augmenting path from s to t . Define $S = \{i \in N \setminus \{n\} \mid i_s \text{ is labeled}\}$, $T_1 = \{j \in N \setminus \{0\} \mid j_t \text{ is unlabeled}\}$, $T_2 = \{j \in T_1 \mid t \text{ can be reached from } j_t \text{ via an augmenting path}\}$, and $T_3 = \{j \in T_2 \mid \text{the flow on } (j_t, t) \text{ in } G''_k \text{ is strictly lower than } u_j\}$. T_3 limits the augmenting path in the definition of T_2 to a single edge. The set of arcs considered for branching is set (S, T_3) if it is not empty, otherwise (S, T_2) if it is not empty, otherwise (S, T_1) (which is never void).

We limit the set of candidate arcs to include only the arcs that have non-zero flow in the remainder network; this set is never empty. Indeed, we mimic the flow in the remainder network with the flow in the partial network: we acknowledge the flow-carrying arcs in the left branch and afterwards destroy feasibility of the partial network flow in the right branch. Choice between eligible arcs is based on the highest sum of flow in G''_k on the arc itself plus the other arcs that are added to Ω_k by transitivity (not constraint propagation). This sum is an estimate of the increase in flow in G''_k that is achieved by the addition of the arc. A tiebreaker rule selects an arc (i, j) with lowest difference $|j - i|$. Multiple other evaluation criteria have been considered but turned out to lead to less efficient results.

8.3.6. Computational experiments

We have implemented the algorithms in C++ using the Microsoft Visual C++ 6.0 programming environment, on a Dell XPS B800r PC with a Pentium III processor. Section 8.3.6.1 explains the general experimental set-up. Different branching schemes are compared in section 8.3.6.2. Objective function comparisons with an allocation heuristic are the subject of section 8.3.6.3.

8.3.6.1. Experimental setup

For various values of n (21, 31, 41 and 61) a set of 200 scheduling instances has been generated by means of RanGen, a recently developed network generator [DEM 03]. The detailed characteristics of these instances are discussed in [LEU 04].

We have mentioned before that any schedule may be the input for the resource allocation algorithm. In our experiments, we use a minimum makespan schedule based on deterministic baseline durations for each activity. The schedule is obtained by means of the branch-and-bound algorithm of Demeulemeester and Herroelen [DEM 97]; the scheduling algorithm is truncated after one minute of CPU-time. We assume that only duration increases occur compared with the baseline plan; Gutierrez and Kouvelis [GUT 91] provide a motivation based on expected worker behavior

under Parkinson's Law. The duration of activity i ($i \neq 0, n$) is disrupted with a probability that is drawn from a uniform distribution on the interval $[0; 0.7]$. In case of disruption, the real duration P_i exceeds the baseline duration p_i by a length $P_i - p_i$, which is a random variable. We assume exponential activity disruption lengths, with average length if disrupted equal to the baseline duration. The cost coefficients c_i are integer values randomly selected from domain $[1; 5]$. We opt for 450, 350, 300 and 250 iterations for $n=21, 31, 41$ and 61, respectively. The final performance evaluation of an allocation after terminating the algorithm is carried out by means of 2,000 simulations.

8.3.6.2. Branching schemes

Table 8.1 presents computational results for the different branching schemes outlined in section 8.3.2: binary branching and branching based on MFSS. A time limit of 150 seconds of CPU-time is imposed. We report the computational results averaged over all instances, solved to guaranteed optimality or not, such that we examine the performance of a truncated branch-and-bound heuristic; the values between parentheses are only for the problems that were solved to optimality. The enumeration of the MFSS is performed as described in [STO 01]. When an MFS consists of two activities, it is already dealt with by the single machine rule and is not listed. We limit the available memory space to 10 MB. For $n = 61$, this storage space was exceeded in 6.5% of the cases (problems marked with an asterisk) and the branch-and-bound tree for the algorithms that use this information was not entered. These problems are considered as "not solved to optimality", and are not taken into account when computing the average values for the corresponding table entries marked by a double asterisk. The MFSS are ordered as in [STO 01], based on the effect on the initial lower bound and on the number of branching alternatives.

For each algorithm, the fraction of the running time consumed by simulation is recorded; this is the larger part of the CPU-time for all runs. Both in number of problems solved to optimality (# optimal) and in terms of computational effort (CPU-time (CPU), number of search nodes (# nodes) and number of objective function evaluations (# eval)), binary branching performs better than MFS-based branching. The top part of Table 8.1 provides some additional data, namely the number of simulations per evaluation (# simul/eval) and some details on the MFS-structure of the problems in the data set. We report the total number of MFSS (# MFSS) as well as the number of MFSS with more than two activities (on the same line, separated by a semi-colon). We notice that almost all MFSS contain more than two activities. We also mention the average computation time to obtain the MFS-information (CPU MFS); these times are no longer negligible for $n = 61$.

		$n = 21$	$n = 31$	$n = 41$	$n = 61$
# simul/eval		450	350	300	250
# MFSS		146; 137	1542; 1523	11323; 11288	209134; 209059*
CPU MFS (s)		0	0.01	0.14	16.13
binary branching	# nodes	1503	11018 (3640)	14680 (4099)	12971 (4717)
	CPU (s)	1.82	22.35 (7.19)	36.85 (9.83)	47.07 (16.57)
branching	# eval	549	5031 (1673)	6756 (1891)	6339 (2337)
% simul	% simul	96.11%	94.90% (94.87%)	95.69% (95.79%)	96.49% (96.36%)
# optimal	# optimal	100%	89.5%	82.5%	77.5%
MFS-	# nodes	5040 (1673)	10846 (2353)	12620 (3367)	17336 (8051)**
branching	CPU (s)	12.96 (4.19)	40.29 (8.17)	67.73 (11.31)	116.42 (30.46)**
	# eval	4105 (1315)	8891 (1853)	9582 (2047)	10540 (4318)**
% simul	% simul	94.02% (93.76%)	92.30% (92.08%)	84.03% (92.36%)	70.76% (90.63%)**
# optimal	# optimal	94%	77.5%	60%	33%

Table 8.1. Results of the two implemented algorithms on the different datasets

8.3.6.3. Comparison with the greedy heuristic

The greedy heuristic (see section 8.2.3) is used to compare the objective function with the binary branching algorithm (again truncated after 150 seconds). The greedy routine is written as “AMR”. The results are summarized in Table 8.2, where we show the deviation between the objective function values obtained by the branch-and-bound algorithm and by AMR. The CPU-time consumed by AMR is negligible. The branch-and-bound performs significantly better than AMR, but evidently requires more computational effort. Inclusion of the problems for which optimality was not guaranteed increases the deviations, which indicates that the hard problems have a wider variety of objective function values, and thus benefit more from more intensive optimization efforts. In Table 8.2 we also provide details on the results obtained for a simulation in which all disruption probabilities as well as the average disruption lengths are divided by 2. We conclude that for smaller and less frequent disruptions, the advantage of a sophisticated resource allocation technique increases, while computational effort remains in the same order of magnitude.

	$n = 21$	$n = 31$	$n = 41$	$n = 61$
avg dev AMR	5.03% 6.25% (5.81%)	5.65% (5.56%)	5.03% (4.51%)	
avg dev AMR, shorter disr	5.89% 7.99% (7.31%)	7.37% (7.18%)	6.44% (5.76%)	

Table 8.2. Improvement in objective function value compared with allocation heuristic AMR

8.4. A polynomial algorithm for activity insertion

In this section, we propose a method to tackle the presence of an unexpected activity, which has to be inserted in the baseline schedule. We consider the case where resource allocation has already been performed (for example, with the algorithm described in section 8.3 or with the heuristic presented in section 8.2.3). For the reasons invoked in sections 8.2.4 and 8.3.1, we wish to modify the existing resource allocation as little as possible, so as to favor system stability. A second objective is to minimize the makespan increase occurring after insertion. Our approach lies in replacing the stability objective by constraints. As explained below, such constraints allow us to find the optimal insertion position in polynomial time.

Section 8.4.1 states the insertion problem and formally defines a feasible insertion. Section 8.4.2 describes how to compute the local impact of a given feasible insertion on the makespan. Section 8.4.3 gives feasibility conditions for an insertion. The concepts of sufficient insertions and insertion cuts are presented in

section 8.4.4. Section 8.4.5 presents dominance rules allowing to discard *a priori* dominated insertions. A polynomial algorithm able to generate a dominant set of feasible insertions is presented in section 8.4.6. Experimental results are presented in section 8.4.7. A more detailed description of the method can be found in [ART 03]. Extension of activity insertion in resource flow networks in the context of an RCPSP with minimum and maximum time lags is studied in [ART 07, ART 08a].

8.4.1. Insertion problem formulation

We consider an RCPSP instance (N, A, a, p, R) as defined in section 8.2.1 with $N = \{0, 1, \dots, n-1, n\}$, where $1, \dots, n-1$ denote the actual project activities and 0 and n denote the dummy start and end activities, respectively. We assume in this section that a resource allocation, represented by a feasible flow f has already been computed, for example with the algorithm described in section 8.3 or with the heuristic presented in section 8.2.3. Baseline schedule \mathcal{S} is one of the minimal makespan schedules compatible with $G_{\chi(f)}$, for instance $\mathcal{S} = \Theta(A \cup \chi(f))$ the earliest start schedule.

We now assume that an unexpected activity, $n+1$, has to be inserted in the project. The activity is defined by its duration \tilde{p}_{n+1} , its resource demand \tilde{R}_{n+1} , its direct predecessors set $\Gamma_{n+1}^- \subseteq N \setminus \{n\}$ and its direct successors set $\Gamma_{n+1} \subseteq N \setminus \{0\}$.

Let $\tilde{p}_i = p_i$, $\tilde{R}_i = R_i$, $\forall i = 0, \dots, n$, $\tilde{N} = N \cup \{n+1\}$ and $\tilde{A} = A \cup (\Gamma_{n+1}^-, n+1) \cup (n+1, \Gamma_{n+1})$. The data associated with this new activity defines a new RCPSP $(\tilde{N}, \tilde{A}, a, \tilde{p}, \tilde{R})$. To maximize the stability of resource allocation, we prevent the initial resource allocation from being deeply modified by the insertion. To this end, the only flow modifications allowed lie in rerouting a part of the existing flow to the inserted activity while sending back the rerouted flow units to the initial destination activities after the completion of the inserted activity. Formally, a solution of the insertion problem can be represented by a matrix q where q_{ij} represents the amount of flow rerouted from the initial flow f_{ij} to activity $n+1$, $\forall i, j \in N \times N$. It follows that any matrix q defines an *insertion*. An insertion q is feasible if and only if flow \tilde{f} , defined by:

$$\tilde{f}_{ij} = f_{ij} - q_{ij} \quad \forall i, j \in N \times N \quad (8.8)$$

$$\tilde{f}_{i(n+1)} = q(i, N) \quad \forall i \in N \quad (8.9)$$

$$\tilde{f}_{(n+1)i} = q(N, i) \quad \forall i \in N \quad (8.10)$$

is a feasible flow for the RCPSP $(\tilde{N}, \tilde{A}, a, \tilde{p}, \tilde{R})$.

The insertion problem amounts to finding an insertion q such that \tilde{f} defined by (8.8)-(8.10) is feasible for $G(\tilde{N}, \tilde{A}, a, \tilde{p}, \tilde{R})$ and gives a schedule $\tilde{\mathcal{S}} = \Theta(\tilde{A} \cup \chi(\tilde{f}))$ of minimal makespan. We assume that $G(\tilde{N}, \tilde{A} \cup \chi(f))$ is acyclic. Otherwise the initial flow and the precedence constraints of the new activity are incompatible and the insertion problem simply has no solution, because of the stability constraints.

8.4.2. Evaluation of a feasible insertion

The makespan of the earliest start schedule corresponding to a feasible insertion q can be computed efficiently by using the earliest start and the latest completion times of the activities in the minimal-makespan schedules compatible with the initial flow f . Let ES_i denote the earliest start time of i according to flow f , i.e. its start time in schedule $\Theta(A \cup \chi(f))$, defined in section 8.2.2. Let LF_i denote the latest completion time of i according to flow f . The latest completion times can be computed recursively by setting $LF_n = C_n$ and $LF_i = \min_{j \in \sigma_{A \cup \chi(f)}(i)} (LF_j - p_j)$ for any other activity i . LF_i is the makespan of $\Theta(A \cup \chi(f))$ minus the tail of i , plus the duration of i . The latest completion times have the following property: if an activity i ends at time $t > LF_i$, then the makespan increases by $t - LF_i$ (provided that the resource allocation described by f is followed). Taking account of equations (8.8)-(8.10), inserting $n + 1$ according to insertion q may only right-shift the direct successors of $n + 1$ in Γ_{n+1} and any activity of set $\sigma(q) = \{j | j \in N, \exists i \in N, q_{ij} > 0\}$. Now if $\pi(q) = \{j | j \in N, \exists i \in N, q_{ji} > 0\}$, the start time increase of activities in $\Gamma_{n+1} \cup \sigma(q)$ comes directly from the start time of $n + 1$ after insertion, noted $\tilde{ES}_{n+1}^{\pi(q)}$. The smallest and latest completion time possibly violated by the insertion is noted $\tilde{LF}_{n+1}^{\sigma(q)}$. The maximum violation of latest completion times, corresponding to the makespan increase, is noted $\Delta C_{\max}(\pi(q), \sigma(q))$. These values are given by:

$$\tilde{ES}_{n+1}^{\pi(q)} = \max_{i \in \Gamma_{n+1}^- \cup \pi(q)} (ES_i + p_i) \quad (8.11)$$

$$\tilde{LF}_{n+1}^{\sigma(q)} = \min_{i \in \Gamma_{n+1} \cup \sigma(q)} (LF_i - p_i) \quad (8.12)$$

$$\Delta C_{\max}(\pi(q), \sigma(q)) = \max(0, \tilde{ES}_{n+1}^{\pi(q)} + p_{n+1} - \tilde{LF}_{n+1}^{\sigma(q)}) \quad (8.13)$$

Assuming that earliest start and latest completion times of activities N are computed in advance, the makespan increase can be evaluated in $\mathcal{O}(n^2)$ for a given feasible insertion q .

EXAMPLE.– Consider the resource flow given in Figure 8.2(a). The corresponding earliest start times are represented in Figure 8.3, i.e. $ES = (0, 1, 0, 2, 0, 3)$. We can also compute the associated latest completion times $LF = (0, 3, 2, 3, 2, 3)$. Now assume that a new activity $n + 1 = 6$ defined by $p_6 = 2$ and $R_6 = 2$ must be inserted between activities 4 and 3 ($\Gamma_6^- = \{4\}$ and $\Gamma_6 = \{3\}$). It is easy to verify that insertion q with non-zero components $q_{02} = 1$ and $q_{41} = 1$ is a feasible insertion. It consists of inserting activity 6 in the flow between activities $\pi(q) = \{0, 4\}$ and $\sigma(q) = \{2, 1\}$. The earliest start time of activity 6 after insertion is $\tilde{LF}_6^{\sigma(q)} = LF_2 - p_2 = 0$. The makespan increase is $\Delta C_{\max}(\pi(q), \sigma(q)) = 3$.

8.4.3. Insertion feasibility conditions

The following property, which simply comes from the definition of an insertion q and from the feasibility of the issued flow \tilde{f} , gives a necessary and sufficient condition of feasibility for an insertion q .

PROPERTY 8.1.– *An insertion q is feasible if and only if*

- (1) $q(N, N) = u_{n+1}$,
- (2) $0 \leq q_{ij} \leq f_{ij}, \forall i, j \in N \times N$,
- (3) *there is no path in $G_{\chi(f)}$ from $\Gamma_{n+1} \cup \sigma(q)$ to $\Gamma_{n+1}^- \cup \pi(q)$.*

Indeed, (1) implies that the incoming and outgoing flow of $(n + 1)$ are equal to its demand. (2) states the remaining flow $\tilde{f}_{ij} = f_{ij} - q_{ij}$ must be non-negative and not larger than its initial value. (3) states that graph $G_{\chi(\tilde{f})}$, issued from the insertion, is acyclic. Part (3) of property 8.1 can be replaced by a sufficient condition which makes it possible to avoid path computations from $\Gamma_{n+1} \cup \sigma(q)$ to $\Gamma_{n+1}^- \cup \pi(q)$. We obtain the following sufficient condition for insertion feasibility:

PROPERTY 8.2.– *An insertion q is feasible if conditions (1) and (2) of property 8.1 are verified and if at least one of the two following conditions is verified:*

- (3a) $\min_{i \in \Gamma_{n+1} \cup \sigma(q)} (ES_i + p_i) > \max_{j \in \Gamma_{n+1}^- \cup \pi(q)} ES_j$,
- (3b) $\min_{i \in \Gamma_{n+1} \cup \sigma(q)} LF_i > \max_{j \in \Gamma_{n+1}^- \cup \pi(q)} (LF_j - p_j)$.

EXAMPLE.– Insertion q for activity 6 in the flow given in Figure 8.2(a), defined by non-zero components $q_{02} = 1$ and $q_{41} = 1$ verifies

$$\min_{i \in \Gamma_6 = \{3\} \cup \sigma(q) = \{2, 1\}} (ES_i + p_i) = 2 > \max_{j \in \Gamma_6^- = \{4\} \cup \pi(q) = \{0, 4\}} ES_j = 0.$$

From property 8.2, q is feasible. A cycle search is not necessary to check its feasibility.

8.4.4. Sufficient insertions and insertion cuts

We call *sufficient insertion* any ordered pair of disjoint activity sets (π, σ) such that $f(\pi, \sigma) \geq R_{n+1}$ and such that there is no path in $G_{\chi(f)}$ from $\Gamma_{n+1} \cup \sigma$ to $\Gamma_{n+1}^- \cup \pi$. As a matter of fact, given such an ordered pair, a feasible insertion q (in the sense given by property 8.1) can be easily computed by the following algorithm. Initialize $q_{ij} = 0$, $\forall i, j \in N \times N$, then update $q_{ij} = \min(R_{n+1} - q(N, N), f_{ij})$ for each $i, j \in N \times N$, while $q(N, N) < R_{n+1}$. By using (8.13), we compute in $\mathcal{O}(n^2)$, the makespan increase $\Delta C_{\max}(\pi, \sigma)$. Since $\pi(q) \subseteq \pi$ and $\sigma(q) \subseteq \sigma$, it holds that $\Delta C_{\max}(\pi(q), \sigma(q)) \leq \Delta C_{\max}(\pi, \sigma)$.

Conversely, consider the optimal insertion q^* giving the minimal makespan increase $\Delta C_{\max}(\pi(q^*), \sigma(q^*))$. Let (π, σ) such that $\pi = \pi(q^*)$ and $\sigma = \sigma(q^*)$. (π, σ) is indeed a sufficient insertion, obviously verifying $\Delta C_{\max}(\pi(q^*), \sigma(q^*)) = \Delta C_{\max}(\pi, \sigma)$. From these two properties, it follows that the search for an optimal insertion can be replaced by the search for a sufficient insertion (π, σ) of minimal $\Delta C_{\max}(\pi, \sigma)$. This formulation has the merit of avoiding the enumeration of the q_{ij} components. As noticed in section 8.3.2, it is not the quantity but the existence of a flow between two activities that matters. With a slight abuse of terminology, we say that an insertion q is *included* in a sufficient insertion (π, σ) if $\pi(q) \subseteq \pi$ and $\sigma(q) \subseteq \sigma$.

We call *insertion cut* a sufficient insertion (π, σ) such that $\pi \cup \sigma = N$. Then, we have $f(\pi, \sigma) = a$, thanks to the properties of flows in a network and by giving a capacity to each arc (i, j) equal to f_{ij} . We can show that for any sufficient insertion (π, σ) , there exists an insertion cut $(\bar{\pi}, \bar{\sigma})$ such that $\pi \subseteq \bar{\pi}$ and $\sigma \subseteq \bar{\sigma}$ with $\Delta C_{\max}(\pi, \sigma) \leq \Delta C_{\max}(\bar{\pi}, \bar{\sigma})$. It suffices to integrate in $\bar{\pi}$ all activities leading to π (including activities of π) by a path issued from 0, as well as all the activities leading to σ except activities of σ . We obtain $\bar{\sigma} = N \setminus \bar{\pi}$. Thus, if we are able to generate a dominant set of insertion cuts and if we have an algorithm to find the best sufficient insertion included in each cut, we are able to find the optimal insertion. The proposed algorithm is based on these principles, and on the dominant conditions given in the next section.

EXAMPLE.– Consider cut (π, σ) with $\pi = \{0, 4\}$ and $\sigma = \{1, 2, 3, 5\}$. (π, σ) is a sufficient insertion since there is no path from $\sigma \cup \Gamma_6$ to $\pi \cup \Gamma_6^-$ and $f(\pi, \sigma) = 3 = a > R_6$. Moreover, since $\Delta C_{\max}(\pi, \sigma) = 3$, any insertion q such

that $\pi(q) \subseteq \pi$ and $\sigma(q) \subseteq \sigma$ verifies $\Delta C_{\max}(\pi(q), \sigma(q)) \leq 3$. This is the case for insertion q of non-zero components $q_{02} = q_{41} = 1$, considered in previous sections.

8.4.5. Insertion dominance conditions

An insertion q_1 dominates another insertion q_2 if

$$\Delta C_{\max}(\pi(q_1), \sigma(q_1)) \leq \Delta C_{\max}(\pi(q_2), \sigma(q_2))$$

A sufficient insertion (π, σ) dominates an insertion q if any insertion ρ verifying $\pi(\rho) \subseteq \pi$ and $\sigma(\rho) \subseteq \sigma$ is such that $\Delta C_{\max}(\pi(\rho), \sigma(\rho)) \leq \Delta C_{\max}(\pi(q), \sigma(q))$, i.e. if the entire family of insertions characterized by (π, σ) dominates q . Given a sufficient insertion, we can discard from the search any dominated insertion.

The following dominance conditions, based on makespan increase expressions (8.11), (8.12) and (8.13), can be established. In properties 8.3, 8.4 and 8.5, (π, σ) denotes a sufficient insertion and q denotes an insertion.

Property 8.3 gives a dominance condition involving two activities i and j .

PROPERTY 8.3.– *If there are two activities $i \in \pi(q)$ and $j \in \sigma(q)$ such that $\tilde{ES}_{n+1}^{\pi} \leq ES_i + p_i$ and $\tilde{LF}_{n+1}^{\sigma} \geq LF_j - p_j$, (π, σ) dominates q .*

Property 8.4 gives a dominance condition of a sufficient insertion (π, σ) on an insertion q considering an activity $j \in \sigma(q)$ if $\tilde{ES}_{n+1}^{\pi} = \max_{k \in \Gamma_{n+1}^-} (ES_k + p_k)$, which corresponds to the case where $n+1$ is inserted immediately after its predecessor with the largest and earliest completion time for any insertion “included” in (π, σ) .

PROPERTY 8.4.– *If $\tilde{ES}_{n+1}^{\pi} = \max_{k \in \Gamma_{n+1}^-} (EF_k + p_k)$ and if there is an activity $j \in \sigma(q)$ verifying $\tilde{LF}_{n+1}^{\sigma} \geq LF_j - p_j$, (π, σ) dominates q .*

In a symmetric way, property 8.5 gives a dominance condition of a sufficient insertion (π, σ) upon an insertion q , considering an activity $i \in \pi(q)$ if $\tilde{ES}_{n+1}^{\pi} = \min_{k \in \Gamma_{n+1}} (LF_k - p_k)$, which corresponds to the case where $n+1$ is inserted immediately before its successor of smallest latest start time for any insertion “included” in (π, σ) .

PROPERTY 8.5.– *If $\tilde{LF}_{n+1}^{\sigma} = \min_{k \in \Gamma_{n+1}} (LF_k - p_k)$ and if there is an activity $i \in \pi(q)$ verifying $\tilde{ES}_{n+1}^{\pi} \leq ES_i + p_i$, (π, σ) dominates q .*

EXAMPLE.– Insertion cut (π, σ) , presented in section 8.4.4, dominates insertion q , presented in section 8.4.2 because $\tilde{ES}_6^{\pi} = ES_4 + p_4 = 1$ with $4 \in \Gamma_6^-$ and activity

$2 \in \sigma(q)$ is such that $\tilde{LF}_6^{\sigma} = LF_2 - p_2 = 0$. Property 8.4 holds. Another example considers insertion cut (π', σ') with $\pi' = \{0, 2, 4\}$ and $\sigma' = \{1, 3, 5\}$. (π', σ') dominates (according to property 8.3) any insertion q' such that $2 \in \pi(q')$ and $1 \in \sigma(q')$. Furthermore, it also dominates any insertion q' such that $2 \in \pi(q')$ from property 8.5 because $\tilde{LF}_6^{\sigma'} = LF_3 - p_3$ with $3 \in \Gamma_6$.

8.4.6. An algorithm for enumerating dominant sufficient insertions

The analysis of the dominance conditions presented in the preceding section shows that we can define a dominant set of sufficient insertions, comprising at most n^2 sufficient insertions. Indeed, notice from property 8.3 that it suffices to compute a single sufficient insertion by ordered pair of activities (i, j) : the one that verifies $\tilde{ES}_{n+1}^{\pi} = ES_i + p_i$ and $\tilde{LS}_{n+1}^{\sigma} = LF_j - p_j$. Properties 8.4 and 8.5 allow us to reduce the cardinality of this set by considering the time window associated with the activity to insert. In [ART 03], an algorithm for generating a series of dominant insertion cuts is proposed. For each generated insertion cut, a dominant sub-series of included sufficient insertions is computed, each one being evaluated. The algorithm allows us to generate and evaluate all the dominant, sufficient insertions in $\mathcal{O}(n^2)$.

8.4.7. Experimental results

To validate the proposed insertion algorithm in a dynamic environment, we consider an initial RCPSP and a schedule computed by a static method. We select the parallel scheduling scheme applied with the MINLFT¹ priority rule [KOL 98] to produce the baseline schedule \mathcal{S} . A feasible flow f , compatible with \mathcal{S} , is then generated using algorithm AMR, presented in section 8.2.3.

In our experimental framework, a single unexpected activity has to be inserted in the project schedule. We test and compare three algorithms for repairing schedule \mathcal{S} . The first one, denoted by PARLFT, consists of solving the new RCPSP instance ignoring the baseline schedule with the above-defined static algorithm (the parallel scheduling scheme with the MINLFT priority rule). The second algorithm, denoted by RESCHINS, consists of inserting the activity in flow f using the algorithm proposed in the preceding section. The third algorithm, denoted by RESCHINSACTIVE, consists of applying RESCHINS plus a post-processing phase in which a schedule is generated using the serial scheduling scheme [KOL 98] and

1. The selected activity is the one with the smallest latest finishing time considering the precedence constraints only.

the following priority rule: at each step of the serial scheduling scheme, select the activity with the smallest start time in $\Theta(\tilde{A} \cup \chi(\hat{f}))$, i.e. the earliest start schedule compatible with the flow issued from the insertion. The post-processing phase aims at filling the resource idle times possibly branded activity right-shifts during insertion. The schedule thus obtained is as least as good as the one obtained after insertion. The three algorithms have the same worst-case time complexity ($\mathcal{O}(n^2)$); the last one is obviously slower than the first two.

We carried out our experiments on the 600 Kolish *et al.* [KOL 95] instances with 120 activities. Since these problems comprise several resources, we used the multi-resource variant of the proposed insertion algorithm, based on the same principles and detailed in [ART 03]. For each of these instances, we generated 27 unexpected activities, each one being inserted in the same baseline schedule and corresponding to different activity parameters. We coded these parameters using a three digit number $p f R$. $p \in \{0, 1, 2\}$ defines the duration of the inserted activity, i.e. small, medium, large. $f \in \{0, 1, 2\}$ describes the width (in an increasing order) of the inserted activity time window, defined by its predecessors and successors. $R \in \{0, 1, 2\}$ defines the demand of the activity by increasing amounts.

In [ART 03], experimental results on the mean and maximum baseline makespan increase, for each of the three algorithms and for each of the 27 characteristics are averaged over the 600 instances. It appears that the level of resource demand is a discriminating factor. For high resource demands, RESCHINS and RESCHINSACTIVE are superior to PARLFT. For small and medium resource demands, PARLFT obtains better mean increases than RESCHINS. This effect is largely tempered by RESCHINSACTIVE, for which the difference with PARLFT becomes insignificant as soon as PARLFT obtains better results. In terms of worst-case evaluation, PARLFT obtains much worse results than RESCHINS and RESCHINSACTIVE since it can obtain dramatic results in terms of maximal makespan increase. The good relative performance of RESCHINSACTIVE shows that it can be fruitful to combine an insertion algorithm with an active schedule generation scheme in a dynamic environment.

8.5. Conclusion

This chapter has examined how to cope with uncertainties in the context of project scheduling with limited resources. We have considered variability in the activity durations and the arrival of new unexpected activities. Our objective in the two cases has been to maintain the stability of the initial schedule, on the one hand by

keeping the initial resource allocation unchanged, and on the other hand by selecting as objective function the expected difference between actual and planned activity starting times. A resource flow model has been used for modeling resource allocation decisions. In the case of activity duration perturbations, our approach is to establish a reactive policy based on resource flows and to find a resource allocation that minimizes the expected starting time deviation. For the second type of uncertainty, a new task is inserted in the pre-computed flow network with an eye to minimizing the makespan increase. Experimental results show that, in both cases, the extra effort for taking uncertainty into account is moderate, while substantial gains in stability and robustness can be achieved.

8.6. Bibliography

- [AHU 93] AHUJA R.K., MAGNANTI T.L. and ORLIN J.B., *Network Flows. Theory, Algorithms and Applications*, Prentice Hall, 1993.
- [ART 00] ARTIGUES C. and 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. and 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 07] ARTIGUES C. and BRIAND C., The resource-constrained activity insertion problem with minimum and maximum time lags, LAAS report no. 07678, LAAS-CNRS, University of Toulouse, 2007.
- [ART 08a] ARTIGUES C. and BRIAND C., “Activity insertion problem in a RCPSP with minimum and maximum time lags”, in ARTIGUES C., DEMASSEY S. and NÉRON E. (Eds.), *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*, p. 171–190, ISTE-Wiley, 2008.
- [ART 08b] ARTIGUES C., DEMASSEY S. and NÉRON E. (Eds.), *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*, ISTE-Wiley, 2008.
- [AYT 05] AYTUG H., LAWLEY M.A., MCKAY K., MOHAN S. and 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.
- [BOW 95] BOWERS J.A., “Criticality in resource constrained networks”, *Journal of the Operational Research Society*, vol. 46, p. 80–91, 1995.
- [BRU 99] BRUCKER P., DREXL A., MÖHRING 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.

- [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 03] DEMEULEMEESTER E., VANHOUCKE M. and HERROELEN W., "A random generator for activity-on-the-node networks", *Journal of Scheduling*, vol. 6, p. 13–34, 2003.
- [DEM 08] DEMEULEMEESTER E., HERROELEN W. and LEUS R., "Proactive-reactive project scheduling", in ARTIGUES C., DEMASSEY S. and NÉRON E. (Eds.), *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*, p. 203–211, ISTE-Wiley, 2008.
- [EDM 72] EDMONDS J. and KARP R.M., "Theoretical improvements in algorithmic efficiency for network flow problems", *Journal of the ACM*, vol. 19, p. 248–264, 1972.
- [FOR 62] FORD L.R.J. and FULKERSON D.R., *Flows in Networks*, Princeton University Press, 1962.
- [FOR 97] FORTEMPS P. and HAPKE M., "On the disjunctive graph for project scheduling", *Foundations of Computing and Decision Sciences*, vol. 22, no. 3, p. 195–210, 1997.
- [FUL 62] FULKERSON D.R., "Expected critical path lengths in PERT networks", *Operations Research*, vol. 10, p. 808–817, 1962.
- [GUT 91] GUTIERREZ G.J. and KOUVELIS P., "Parkinson's law and its implications for project management", *Management Science*, vol. 37, p. 990–1001, 1991.
- [HAG 88] HAGSTROM J.N., "Computational complexity of PERT problems", *Networks*, vol. 18, p. 139–147, 1988.
- [HER 98] HERROELEN W., REYCK B.D. and DEMEULEMEESTER E., "Resource-constrained project scheduling: a survey of recent developments", *Computers and Operations Research*, vol. 25, no. 4, p. 279–302, 1998.
- [HER 04] HERROELEN W. and LEUS R., "On the construction of stable project baseline schedules", *European Journal of Operational Research*, vol. 156, no. 3, p. 550–565, 2004.
- [HER 05] HERROELEN W. and LEUS R., "Project scheduling under uncertainty, survey and research potentials", *European Journal of Operational Research*, vol. 165, no. 2, p. 289–306, 2005.
- [IGE 83a] IGELMUND G. and RADERMACHER F.J., "Algorithmic approaches to preselective strategies for stochastic scheduling problems", *Networks*, vol. 13, p. 29–48, 1983.
- [IGE 83b] IGELMUND G. and RADERMACHER F.J., "Preselective strategies for the optimization of stochastic project networks under resource constraints", *Networks*, vol. 13, p. 1–28, 1983.
- [KOL 95] KOLISCH R., SPRECHER A. and DREXL A., "Characterization and generation of a general class of resource-constrained project scheduling problem", *Management Science*, vol. 41, p. 1693–1703, 1995.

- [KOL 98] KOLISCH R. and HARTMANN S., “Heuristic algorithms for the resource-constrained project scheduling problem: classification and computational analysis”, in WEGLARZ J. (Ed.), *Project Scheduling: Recent Models, Algorithms and Applications*, p. 147–178, Kluwer Academic Publishers, 1998.
- [KOL 01] KOLISCH R. and PADMAN R., “An integrated survey of deterministic project scheduling”, *Omega*, vol. 49, no. 3, p. 249–272, 2001.
- [LEU 03] LEUS R., Stability and resource allocation in project planning. Complexity and exact algorithms, PhD Thesis, KU Leuven, 2003.
- [LEU 04] LEUS R. and HERROELEN W., “Stability and resource allocation in project planning”, *IIE Transactions*, vol. 36, no. 7, p. 667–682, 2004.
- [MEH 98] MEHTA S.V. and UZSOY R.M., “Predictable scheduling of a job shop subject to breakdowns”, *IEEE Transactions on Robotics and Automation*, vol. 14, no. 3, p. 365–378, 1998.
- [MÖH 85] MÖHRING R.H., “Algorithmic aspects of comparability graphs and interval graphs”, in RIVAL I. (Ed.), *Graphs and Orders*, Reidel Publishing Company, 1985.
- [MÖH 89] MÖHRING R.H. and RADERMACHER F.J., “The order-theoretic approach to scheduling: the stochastic case”, SLOWINSKI R. and WEGLARZ J. (Eds.), *Advances in Project Scheduling*, Chapter III.4, Elsevier, 1989.
- [MÖH 00] MÖHRING R.H., Scheduling under uncertainty: bounding the makespan distribution, Working Paper no. 700/2000, Department of Mathematics, TU Berlin, Germany, 2000.
- [NAE 89] NAEGLER G. and SCHOENHERR S., “Resource allocation in a network model – the Leinet system”, in SLOWINSKI R. and WEGLARZ J. (Eds.), *Advances in Project Scheduling*, Chapter II.8, Elsevier, 1989.
- [NEU 02] NEUMANN K., SCHWINDT C. and ZIMMERMANN J., *Project scheduling with Time Windows and Scarce Resources*, Lecture Notes in Economics and Mathematical Systems, Springer, 2002.
- [ROY 64] ROY B. and SUSSMAN B., Les problèmes d’ordonnancement avec contraintes disjonctives, Note DS no. 9 bis, SEMA, Paris, 1964.
- [STO 01] STORK F., Stochastic resource-constrained project scheduling, PhD Thesis, TU Berlin, 2001.
- [WU 93] WU S.D., STORER H.S. and CHANG P.C., “One-machine rescheduling heuristics with efficiency and stability as criteria”, *Computers and Operations Research*, vol. 20, no. 1, p. 1–14, 1993.

Chapter 9

Constraint-based Approaches for Robust Scheduling

9.1. Introduction

By constraint-based approach, we refer in this chapter to studies focusing on the representation and processing of constraints in scheduling. These approaches compare and combine operations research methods with constrained representation and processing from artificial intelligence (mainly constraint propagation algorithms). The goal is to design tools to facilitate interaction between models and decision makers, by integrating useful analysis methods in the context of decision support.

In accordance with the classification presented in Chapter 1, the scheduling methods we propose in this chapter are said to be *proactive/reactive*. Indeed, such methods aim to characterize a flexible group of schedules which can be used to face the contingencies that arise during the plan execution, rather than a unique solution [HER 05]. Since flexibility and quality of a set of solutions are conflicting measures (see Chapter 9), we assume that the decision support can realize a satisfactory trade-off.

Three main origins of flexibility are invoked here. The first concerns the *sequential flexibility*, i.e. the possibility of modifying the execution order of tasks on resources. The second concerns the *flexibility over allocation* of resources to tasks when several

Chapter written by Cyril BRIAND, Marie-José HUGUET, Hoang Trung LA and Pierre LOPEZ.

choices of resources (alternatives) are possible to process a task. The third is the *temporal flexibility* associated with the slack of each task to be processed which can be used without degrading the solution quality.

Seeking a trade-off between flexibility and quality, the flexibility of a set of solutions must be measured. An obvious measure is to count the number of different schedules (without enumerating them) amongst a set of solutions. Other metrics can also be used.

This chapter presents how to characterize, avoiding the pitfalls of enumeration, a flexible set of solutions using *interval structures* and *necessary, sufficient and dominant conditions* of feasibility or optimality. Such conditions restrain the possible allocations of resources to tasks, the time windows to perform the tasks, as well as the possible sequencings between tasks. For example, for the latter case, we can define an order only over a subset of the problem tasks. This *partial order* allows us to bound the quality of the characterized flexible set in the worst case, and then to measure its flexibility. Moreover certain orders can be rather insensitive to data variations; this characteristic can be particularly useful in the context of robust scheduling.

First, we define the concepts of necessary, sufficient and dominant conditions, as well as that of partial order which we will use in what follows. Then we address the notion of interval structure and several related concepts. Finally, we will focus on the study of necessary conditions of feasibility, and sufficient and dominant conditions of optimality.

9.2. Necessary/sufficient/dominant conditions and partial orders

In scheduling, the concepts of necessary and sufficient conditions are related to either the feasibility of a problem or the optimization of a criterion. Considering feasibility, the theory of *constraint-based analysis* [ERS 76] provides necessary and sufficient conditions that *all* the solutions must fulfill according to the problem constraints. Owing to its NP-complete feature, constraint-based analysis splits into the search for necessary conditions on the one hand and for sufficient conditions on the other hand. The sets of solutions satisfying *necessary* conditions consist of all the feasible (resp. optimal, considering a given criterion) schedules and other unfeasible (resp. non-optimal) schedules. The search for *sufficient* conditions aims at characterizing a subset of all feasible (resp. optimal) schedules.

Other approaches characterize a set of schedules according to *dominant* conditions in relation to optimality or feasibility. When studying feasibility, we say

that a sequence Seq_1 dominates a sequence Seq_2 iff Seq_2 feasible implies Seq_1 feasible [ERS 83]. The dominant sequences can be considered as potentially better than others since if no feasible dominant sequence exists then no solution exists for the global problem.

In the optimization framework, the definition of dominance involves the notion of circuit in conjunctive graphs. Let $G(X, U)$ be a conjunctive graph where X is the set of tasks to be scheduled and U the set of conjunctive arcs between two tasks of the problem [ROY 70]. Let us now consider two conjunctive graphs $G_1(X, U_1)$ and $G_2(X, U_2)$ that describe two sequences Seq_1 and Seq_2 . We state that sequence Seq_1 dominates sequence Seq_2 iff the length of every circuit of $G_2(X, U_2)$ is greater than or equal to the length of the longest circuit of $G_1(X, U_1)$ [FON 80].

The notion of *partial order* is also important. A partial order P is characterized by a pair $P = (X, \preceq_P)$ where every relation \preceq_P over $X \times X$ is reflexive, antisymmetric and transitive. A partial order $P = (X, \preceq_P)$ is a total order if for every couple $(u, v) \in X \times X$ the relations $u \preceq_P v$ or $v \preceq_P u$ are satisfied. A partial order $Q = (X, \preceq_Q)$ is an extension of the partial order $P = (X, \preceq_P)$, if $u \preceq_P v$ implies $u \preceq_Q v$ for every couple $(u, v) \in X \times X$.

As we will see later, partial order relations often correspond to necessary, sufficient or dominant conditions of feasibility (or optimality). By extension, we also speak of necessary, sufficient and dominant partial order.

9.3. Interval structures, tops, bases and pyramids

An interval structure can be defined by a couple $< I, C >$ with $I = \{i_1, \dots, i_n\}$ a set of intervals and C a set of constraints over $I \times I$. Each interval i_j is defined by its lower and upper bounds x_j and y_j . Any constraint between two intervals i_j and i_k can be expressed either by specifying a total order relation among the lower and upper bounds of the intervals or by directly using the relations of the algebra proposed by Allen [ALL 81] (see Figure 9.1).

Top and *base* are particular intervals of an interval structure that can be defined on the basis of Allen's relations.

DEFINITION 9.1.— *A top of an interval structure $< I, C >$ is an interval $t \in I$ such that $\forall i \in I$ the Allen's relation $dur(i, t)$ never holds.*

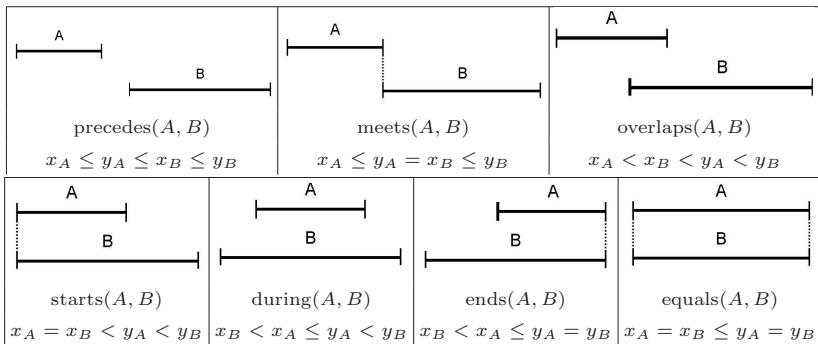


Figure 9.1. Allen's relations

DEFINITION 9.2.—A base of an interval structure $\langle I, C \rangle$ is an interval $b \in I$ such that $\forall i \in I$ the Allen's relation $\text{during}(b, i)$ never holds.

These notions of top and base can be respectively used to define the notions of *t-pyramid* and *b-pyramid* [ESQ 99].

DEFINITION 9.3.—Given a top t_α , a *t-pyramid* P_α related to t_α is the set of intervals $i \in I$ such that $\text{during}(t_\alpha, i)$ holds.

DEFINITION 9.4.—Given a base b_α , a *b-pyramid* P_α related to b_α is the set of intervals such that $\text{during}(i, b_\alpha)$ holds.

For illustration of definitions 3 and 4, let us consider the interval structure of Figure 9.2. It has three tops $\{C, D, E\}$ and four bases $\{A, B, F, G\}$. The involved *t-pyramids* are $P_C = \{B, A, G\}$, $P_D = \{G\}$ and $P_E = \{F, G\}$, and the *b-pyramids* are $P_A = \{C\}$, $P_B = \{C\}$, $P_F = \{E\}$ and $P_G = \{C, D, E\}$.

9.4. Necessary conditions for a generic approach to robust scheduling

9.4.1. Introduction

Constraint-based approaches (also referred to as *constraint programming*) makes a clear distinction between: the constraints model of a problem; analysis methods; and resolution methods. The formalism of *constraint satisfaction problems (CSP)* is used to write the model as a set of variables, a set of domains of values for each variable, and a set of constraints. In the CSP formalism, a constrained variable can also be used to represent an optimization criterion. For example, to iteratively

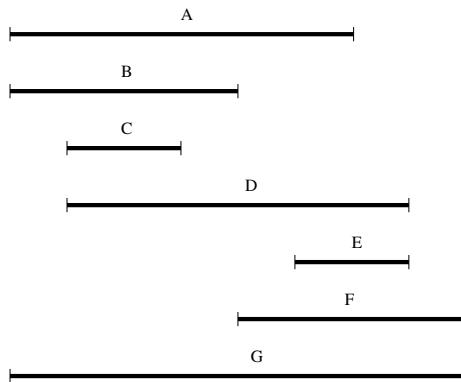


Figure 9.2. An interval structure

minimize an objective function, we can write *criterion_expression* < *criterion_value* where *criterion_expression* is a combination of problem variables and *criterion_value* is a worst-case measure of characterized solutions which decreases at each step. Therefore, the latest value of the criterion is a lower bound on the optimum.

Constraint propagation mechanisms allow us to decide whether there exist feasible solutions for a problem without any enumeration. Such mechanisms characterize a solution space using a logical deduction process to express necessary conditions for feasibility. Thus, any solution removed from the space is effectively inconsistent; the propagation mechanisms are said to be *sound*. Conversely, such mechanisms are also said to be *incomplete* (in the general case) since any solution which belongs to the space has not been proved to be inconsistent, although it may be. However, for certain problems such as *simple temporal problems (STP)* [DEC 91, DEC 03], we know the necessary and sufficient conditions of feasibility. In this particular case, the space of obtained solutions exactly corresponds to the space of feasible solutions.

Generally, constraint propagation mechanisms may either eliminate inconsistent values from variable domains or highlight new constraints between variables. They can also, in certain cases, prove the global inconsistency of the problem before any resolution attempt. This domain filtering provides a more precise characterization of the problem under study and then avoids many doomed resolution attempts.

Constraint propagation satisfies robustness requirements. First, removing the values not belonging to any solution, it provides a set of solutions without any enumeration. The performance of propagation mechanisms may be measured through their impact on domain reduction. Second, it is well-suited to dynamic and reactive

changes of the model. For example, side constraints are not always formally modeled (ill-known parameters, context-dependent behavior) albeit they may be of great impact in practice. This can lead to the addition or the deletion of one or more constraints. In the first case (addition), the solutions already obtained have to be reconsidered whilst the deductions coming from propagation can be kept. In the second case (deletion), the deductions must be reconsidered and the solutions can be kept (as a solution of the original problem is still a solution of the relaxed problem).

9.4.2. Scheduling problems under consideration

In the literature, propagation techniques exist for generic constraints applied to a wide range of scheduling problems. In this chapter, we limit our study to non-preemptive and disjunctive problems. A set of tasks T is to be processed with no interruption on a single resource chosen among a set M of non-sharable resources.

For every task $i \in T$, let st_i be its start time, ft_i its finish time and m_i the resource used to process the task. To model a problem we then used a set $X = \{st_i, ft_i, m_i\}_{i \in T}$ of decision variables and a set C of time and resource constraints linking tasks $i \in T$ and resources $k \in M$.

Set C consists first of constraints on variable domains: $st_i \in [\underline{st}_i, \bar{st}_i]$, $ft_i \in [\underline{ft}_i, \bar{ft}_i]$ and $m_i \in M_i$, where M_i corresponds to the set of resources able to process a task i (in practice, one single resource of M_i will be allocated to process the task). Next, we consider the precedence constraints between tasks: the constraint denoted by $i \prec j$ means task i precedes task j and is represented by the inequality $st_j \geq ft_i$. The generalized precedence constraints may also be taken into account. These can be written as $x_j - x_i \geq a_{ij}$, where x_i and x_j are temporal variables and a_{ij} is an integer value (positive or negative). In addition, there are constraints to represent the disjunctive nature of resources. Every pair of tasks (i, j) using a same resource k must be sequenced: $(i \prec j) \vee (j \prec i)$.

Finally, we have duration constraints. In general, the processing time $p_{i,k} = ft_i - st_i$ of a task i is dependent on the resource $m_i = k$ chosen among M_i . For a given resource k , the processing time may be set (for example $p_{i,k} = 4$) and only one variable (start or finish time) is sufficient to locate the task in time. This processing time can also belong to an interval of possible values $[\underline{p}_{i,k}, \bar{p}_{i,k}]$ (for example $p_{i,k} \in [2, 5]$) and both variables (start and finish times) are necessary to locate this task. Such an interval of values associated with the processing time of a task may, for instance, correspond to an imprecise duration on a given resource. This is interesting

in the context of robustness. To take account of the various choices for allocation, the interval of values for the processing time of a task is defined as the union of intervals for each choice: $[\underline{p}_i, \bar{p}_i] = [\min_{k \in M_i} \underline{p}_{i,k}, \max_{k \in M_i} \bar{p}_{i,k}]$.

9.4.3. Necessary feasibility conditions

For the scheduling problems under consideration, the propagation techniques of the literature allow the deduction of the following necessary feasibility conditions:

1) *time variable domain filtering*: start or finish times of tasks. Often, the associated adjustments do not create holes in the domains of variables. They only lead to an increase in the lower bound and a decrease in the upper bound of the time variable domains;

2) *forbidden allocations*, i.e. domain reductions for allocation variables m_i of tasks $i \in T$; for example $l \notin M_i$;

3) *partial orders*:

- *mandatory precedences between a task i and a set of tasks S* : $i \prec S$ or $S \prec i$;
- *forbidden precedences between a task i and a set of tasks S* : $i \not\prec S$ or $S \not\prec i$;
- *non-insertability of a task i in a set of tasks S* : $i \nmid S$;

4) *distances between time variables*; for example: $ft_j - st_i \geq d_{\min}$ (i.e., a minimum distance constraint), $\underline{p}_i \leq ft_j - st_j \leq \bar{p}_i$ (i.e., a duration constraint), or even constraints which come down partial orders between tasks, such as $st_i \geq ft_j$ (i.e., a mandatory sequencing $j \prec i$) or $ft_j > st_i$ (i.e., a forbidden sequencing $i \not\prec j$, that is $j \prec i$ since the problem is disjunctive).

The necessary conditions presented above are new constraints which enrich the initial set of constraints. The addition of such constraints is different according to their type:

– time variable domain filtering, distances, forbidden allocations and mandatory precedences between tasks are directly added to the set of problem constraints;

– partial orders are added using the provided adjustments, that is:

$$\text{if } S \prec i \text{ then } \underline{st}_i \leftarrow \max \left(\underline{st}_i, \max_{S' \subseteq S} \left(\min_{x \in S'} \underline{st}_x + \sum_{x \in S'} \underline{p}_x \right) \right),$$

$$\text{if } i \prec S \text{ then } \bar{ft}_i \leftarrow \min \left(\bar{ft}_i, \min_{S' \subseteq S} \left(\max_{x \in S'} \bar{ft}_x - \sum_{x \in S'} \bar{p}_x \right) \right),$$

$$\begin{aligned} \text{if } i \not\prec S \text{ then } \underline{st}_i &\leftarrow \max \left(\underline{st}_i, \min_{x \in S} (\underline{st}_x + \underline{p}_x) \right), \\ \text{if } S \not\prec i \text{ then } \overline{ft}_i &\leftarrow \min \left(\overline{ft}_i, \max_{x \in S} (\overline{ft}_x - \underline{p}_x) \right); \end{aligned}$$

We note that the non-insertability of a task in a set is generally used in addition to not-first/not-last conditions in order to deduce mandatory precedences between a task and a set:

$$\text{if } i \nmid S \text{ and } i \not\prec S \text{ then } S \prec i; \quad \text{if } i \nmid S \text{ and } S \not\prec i \text{ then } i \prec S.$$

From an initial set of constraints C , each necessary condition defines a new set of constraints C' including C . Once these necessary feasibility conditions are obtained, the propagation process can be iterated over the new set of constraints until no more deduction can be obtained or an inconsistency arises. At the end of the propagation, we have: for each time variable and for each allocation variable, a domain of values which have not been proved as inconsistent; for each resource, a partial order on the task processing (set of sequences not proved to be inconsistent).

Note that the iterative addition of induced constraints by propagation leads to difficulties on the convergence of deduction towards a unique fixed-point (obtained independently of the order of propagations and the associated algorithms to their processing). This difficulty has been raised in [TOR 00a] and a theoretical framework has been proposed in [DOR 00] to formalize the related algorithms.

9.4.4. Propagation mechanisms

The previous section presents the different types of necessary conditions obtained by constraint propagation techniques in scheduling. We now describe in which cases these conditions can be established. For this purpose, we distinguish time constraint propagation and resource constraint propagation [ESQ 08].

9.4.4.1. Time constraint propagation

The mechanisms of time constraint propagation are particularly important since some main deductions presented previously can be considered as temporal constraints. For *simple temporal problems* (STP) [DEC 91], the propagation mechanisms are sound and complete, that is, the set of feasible solutions is characterized in a necessary and sufficient way. An STP only consists of a conjunctive set of domain constraints and binary constraints, each of them representing a distance

between two times. It can be modeled with a distance graph so that longest-path algorithms allow the characterization of the set of solutions. Traditionally, we use an algorithm checking the *2-consistency* (Bellman-Ford, AC-3, etc.) to characterize the domains of decision variables or an algorithm for *3-consistency* (Floyd-Warshall, PC-2, etc.) to make not only the domains of variables explicit but also all the implicit constraints between each pair of variables.

For general temporal problems, the propagation algorithms are no longer complete and we know only necessary conditions for feasibility of solutions. Here are some examples of general problems for which disjunctions exist between time constraints:

- a task i may start either between [2, 4] or between [7, 9];
- at least a task among $\{i_1, i_2, \dots, i_k\}$ must precede task j ;
- either task i precedes task j , or task j precedes task i .

To represent general temporal problems we need particular graphs such as *temporal constraint networks* (TCN) [DEC 91, SCH 98] or *time-bounds on node graphs* [ESQ 92, ESQ 95] depending on the type of disjunctive constraint we handle.

Let us note finally that there are also some results for ill-known task durations or start/finish times (*contingencies*). This refers to simple temporal problems with uncertainties (STNU). The associated propagation algorithms proposed in [MOR 01], sound and complete, guarantee that every partial solution may be extended whatever the values of the uncertain uncontrollable quantities.

9.4.4.2. Resource constraint propagation

Following the seminal works of Carlier and Erschler [CAR 75, ERS 76], resource constraint propagation techniques are now commonly used, especially inference rules concentrated on a single machine (the so-called *local operations*). The research on this topic has been very active and new rules and efficient algorithms are now available. Among the most famous of them, the *immediate selections* allow the deduction of mandatory precedences between tasks and sets of tasks [CAR 89]. Embedded efficiently in a dedicated branch-and-bound procedure for job-shop scheduling, the immediate selections were able to close the famous instance FT10 [FIS 63], which remained open for more than 20 years.

More recently, *global operations*, also called *shaving*, provided stronger deductions [CAR 94, MAR 96]. The principle of shaving techniques is as follows: 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. Such a technique can be very time-consuming. Nevertheless, it also offers a very promising approach. For example, it allows us to solve instance FT10 to optimality at the root of a search tree [PÉR 96]. For more difficult instances, shaving techniques resulted in a dramatic reduction of expanded nodes (a factor of 10 000) with a factor 4 decrease in CPU time [PHA 00]. In addition, we can consider a hybrid use of shaving type propagation techniques and local search procedures (notably the tabu search method) to enable the efficient extraction of local optima and solve large job-shop problems (15 tasks – 10 machines and beyond) [TOR 00b].

These excellent results confirm the power of constraint-based approaches for scheduling and still stimulate the work in this field. New rules and propagation algorithms have been proposed (see for example [BRU 94, CAS 94, MAR 96]). In addition, the extension of the propagation techniques, other types of constraints (preemptive or cumulative [BAP 01, LOP 91, NUI 94, CAS 96, CAR 00]) enforce their reusability and their effective applicability in a real-life context. *Energetic reasoning* [LOP 92] or the *Jackson pseudo-preemptive schedule* (JPPS) [CAR 04] are some examples.

9.4.5. Interval structures for propagation

The constraint propagation rules traditionally used in scheduling are based on subsets of tasks satisfying particular properties (see the above sections). The search for relevant subsets of tasks to apply such propagation rules to is generally highly combinatorial. In order to reduce this complexity, it is worth examining the temporal structure of the scheduling problem. In this section, we propose interval structures as a support to implement propagation rules in an exhaustive (all the possible deductions have been obtained) and generic (unicity of fixed-point) way.

9.4.5.1. Rank-interval based structures

The set of feasible locations is represented by a *rank interval*. For each task i , the rank interval $I^R(i)$ represents the set of permitted locations of i in a sequence [LOP 98]. With this definition, two simple notions have been introduced (we refer here to Allen's algebra introduced in section 9.3):

- *Equal Rank-Interval Sets* (ERIS) group together tasks having the same rank intervals:

$$i, j \in \text{ERIS } G^E \text{ iff equals } (I^R(i), I^R(j));$$

– *Overlapped Rank-Intervals Set* (ORIS) group together tasks having common rank values:

$i, j \in \text{ORIS } G^O$ iff there exists a series of tasks $k_0 = i, \dots, k_y = j$ such that

$$\forall z = 1, \dots, y, \text{ we obtain } \begin{cases} \text{neither precedes } (I^R(k_{z-1}), I^R(k_z)) \\ \text{nor precedes } (I^R(k_z), I^R(k_{z-1})). \end{cases}$$

The partition into ORISs is attractive as far as the problem is well structured enough so that independent subproblems can be pointed out (in practice only one ORIS can be found). ERISS often lead to a too thin decomposition (small-size groups highly dependent on each other).

A third notion is that of *Included Rank-Intervals Set* (IRIS):

DEFINITION 9.5.– $i, j \in \text{IRIS } G^I$ iff:

includes($I^R(i), I^R(j)$) or

includes($I^R(j), I^R(i)$) or

$\exists x \in G^I, x \neq i, j, \text{ such that during}(I^R(i), I^R(x)) \text{ and during}(I^R(j), I^R(x))$.

where the relation includes takes its value in {starts, during, ends, equals} Allen's relations.

The three notions are illustrated in Figure 9.3. IRISs group together tasks for which there exists an inclusion between rank intervals. IRISs offer the best trade-off between set interdependence of ORISs and activity permutability of ERISSs. Hence the characterization stage is going to be processed over each IRIS to favor computation time savings.

The decomposition in IRISs has been used in disjunctive scheduling problems such as single machine and flow-shop problems. This structuring is close to b-pyramids and is easy to implement. Indeed, the complexity of decomposition in IRISs is that of sorting a set of tasks [LEV 99]. This decomposition allows us to make explicit necessary feasibility conditions in terms of rank adjustments. In addition, it provides a guide for the application of certain propagation rules on subsets of reduced size. For example, the mandatory precedences between a task and a set of tasks (immediate selections) are searched only in IRISs and no longer in the whole set of tasks of the problem tasks.

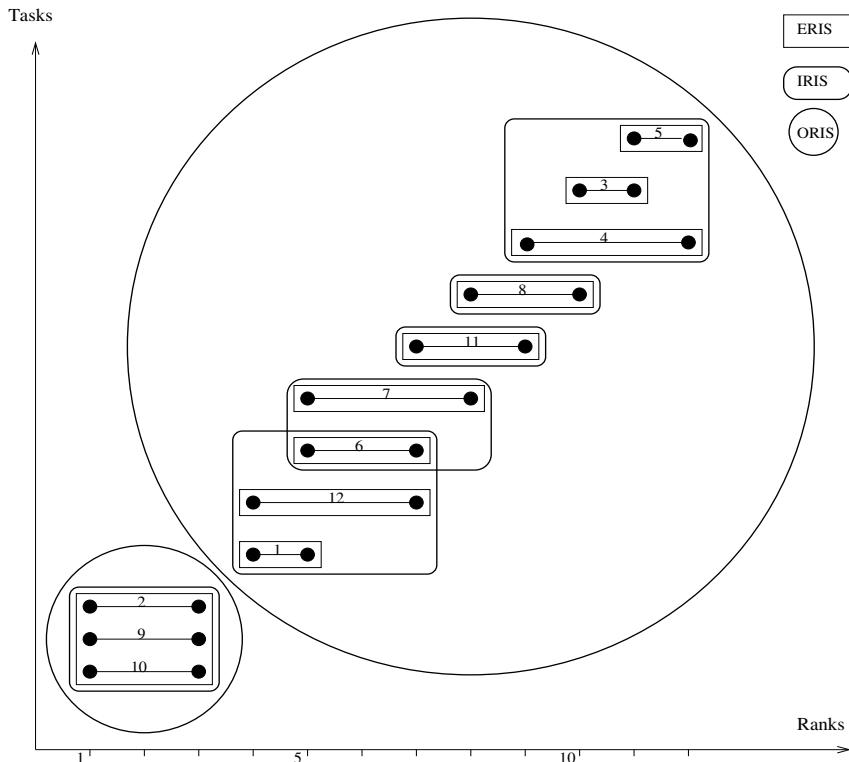


Figure 9.3. Example of groups of tasks

9.4.5.2. Task-interval based structures

A *task interval* groups together a set of tasks in conflict for the use of a resource. The names of two of these conflicting tasks (possibly the same) bound the task interval [CAS 94].

DEFINITION 9.6.– Let i and j be two tasks (eventually $i = j$) using a same resource k and such that $\underline{st}_i \leq \underline{st}_j$ and $\overline{ft}_i \leq \overline{ft}_j$; the task interval $[i, j]$ corresponds to the set of tasks x using k and such that $\underline{st}_i \leq \underline{st}_x$ and $\overline{ft}_x \leq \overline{ft}_j$.

Let $I(T) = \{\underline{st}_i, \overline{ft}_i\} / i \in T\}$ be the set of time windows of the set of tasks T .

We define the *lattice of task intervals*, in short LTI, as the set of task intervals of $I(T)$ ordered by the immediate inclusion relation. An LTI presented without transitive relations can be depicted by an *Hasse diagram* [TOR 99]. Figure 9.5 represents the

Hasse diagram of the set inclusion relation between the task intervals of the example in Figure 9.4.

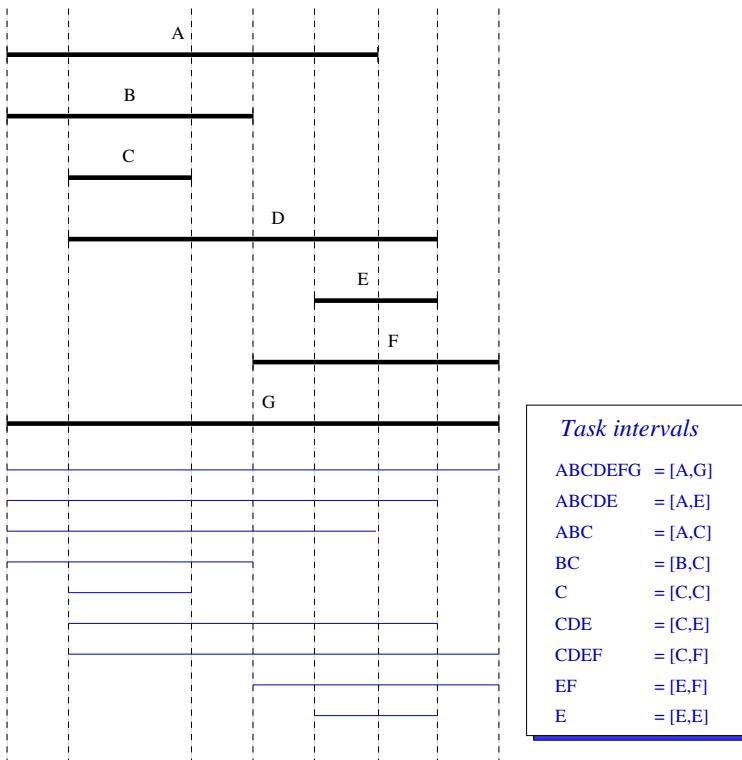


Figure 9.4. Time windows and associated task intervals

To make a link with the definitions of section 9.3, we can notice the following points when characterizing a problem using pyramidal structures:

- 1) the minimal elements of the LTI are tops;
- 2) b-pyramids are particular task intervals: there exists a task (the base) of which the interval strictly includes all the other intervals of the pyramid.

Propagation rules based on the relative location of a set of tasks S and a task $i \notin S$ (edge-finding for example) can be implemented considering *lattices of task intervals* (LTI) [TOR 99]: only the sets of tasks that can have an impact applying these rules are to be generated. Indeed, $i \notin S$ implies $\underline{s}t_i < \min_{x \in S} \underline{s}t_x$ or $\overline{f}t_i > \max_{x \in S} \overline{f}t_x$.

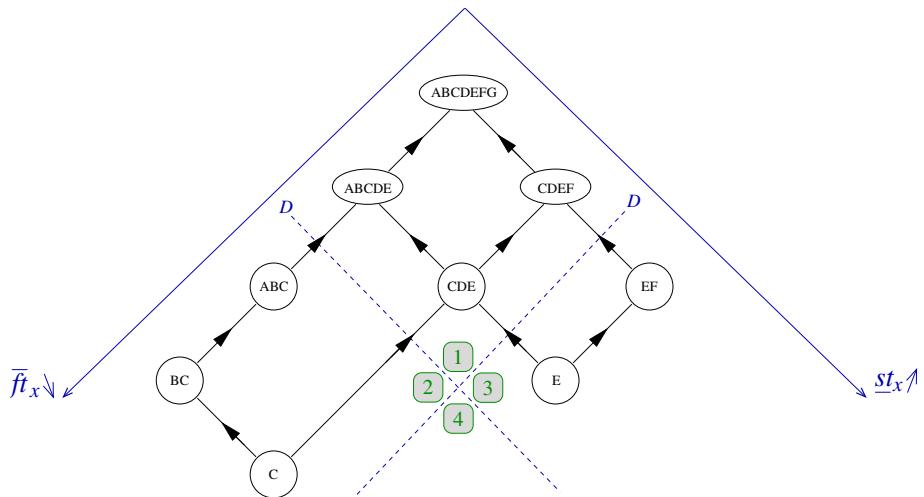


Figure 9.5. Lattice of task intervals: Hasse diagram and areas

In a lattice of task intervals ordered following the non-decreasing release order dates and the non-increasing order of the deadlines, the two previous conditions divide the LTI into four distinctive areas:

- Area ① groups together all the task intervals such that $\underline{st}_i > \min_{x \in S} \underline{st}_x$ and $\bar{ft}_i < \max_{x \in S} \bar{ft}_x$; therefore they include x ; hence this area is irrelevant to trigger the rules.
- On task intervals S of Area ②, if $i \nmid S$ is deduced, since $\bar{ft}_i > \max_{x \in S} \bar{ft}_x$, it yields $S \prec i$.
- On task intervals S of Area ③, if $i \nmid S$ is deduced, since $\underline{st}_i < \min_{x \in S} \underline{st}_x$, it yields $i \prec S$.
- In Area ④, both conditions $\underline{st}_i < \min_{x \in S} \underline{st}_x$ and $\bar{ft}_i > \max_{x \in S} \bar{ft}_x$ are satisfied. If $i \nmid S$ is deduced, it yields $(i \prec S) \vee (S \prec i)$.

Figure 9.5 then illustrates the partition of the LTI for task D . Searching for deductions on D leads us to consider only task intervals located in Areas ② and ③.

9.4.6. Discussion

Determining necessary feasibility conditions has now proved to be very useful for representing and solving scheduling problems. Numerous publications (see for example [DOR 00, BAP 01, BAP 02, BRU 02, LAB 03, LOP 03]) and dedicated sessions in conferences or workshops (IJCAI, AAAI, ECAI, ICAPS,

INAP, CP-AI-OR, etc.) on the topic can be easily found. These approaches use general techniques of constraint satisfaction problems such as propagation, but also solving strategies tuned for scheduling problems: variable/value ordering heuristics, backtracking, etc.

9.5. Using dominance conditions or sufficient conditions

Unlike the study presented in the previous section, there does not exist any generic sufficient condition of optimality or feasibility which can be used for solving any kind of scheduling problem. Nevertheless, there exists a generic dominance condition: it is well known that the search for a feasible solution, or an optimal solution (with regard to a regular criterion), can be restricted to the set of the semi-active schedules. Of course this set is very large and it is difficult to determine its performance. This is why, in order to find tighter sufficient or dominance conditions which characterize a smaller search space, researchers consider the deep nature of a schedule problem by making many assumptions on its structure and properties (number of resources, job-shop, flow-shop, open-shop, equal processing times, task preemption allowed, etc.).

This is precisely what we are going to do in this section by taking an interest in two famous scheduling problems: the single machine scheduling problem with execution windows (the objective function being to minimize the maximum lateness L_{\max}) and the two-machine flow-shop problem (the objective function being to minimize the makespan). For these problems, denoted in the literature as $1|r_j|L_{\max}$ and $F2|prmu|C_{\max}$ respectively, we intend to show how interval structure analysis can be used in order to establish specific dominance or sufficient conditions of optimality. The major value of such conditions is that they characterize, for the considered problems, a set of dominant sequences and a set of optimal sequences respectively, with a known cardinality and performance. Such a set can be used inside a robust scheduling approach, which uses sequential flexibility as a way of dealing with uncertainties.

9.5.1. The single machine scheduling problem

In this section, we define a partial order based on a dominance theorem which was issued in the early 1980s [COU 79, ERS 82, ERS 83]. The dominance, as it is considered by the authors, is related to the feasibility of a schedule. In any case, it has been demonstrated that their dominance theorem is still valid when optimization criterion such as L_{\max} or T_{\max} is considered. The hypothesis frame studied by the authors only takes into account the relative order of the release dates r_j and due dates

d_j of the jobs to be sequenced. Therefore, the processing time p_j as well as the explicit values of r_j and d_j are not used. In other words, whatever the values of r_j , d_j and p_j , the following results are valid as long as the relative order of the release and due dates is kept unchanged.

An interval structure $\langle I_V, C_V \rangle$, associated with a problem V , contains interval $i_j = [r_j, d_j] \in I_V$ for each job $j \in V$. To characterize a dominant set of sequences, the authors use the notions of tops and t-pyramids related to $\langle I_V, C_V \rangle$.

It is assumed that the tops are indexed according to the ascending order of their release dates or, in case of equality, according to the ascending order of their due dates. When both their release dates and due dates are equal, the tops are indexed in an arbitrary order. Thus, if t_α and t_β are two tops then $\alpha < \beta$ if and only if $(r_{t_\alpha} \leq r_{t_\beta}) \wedge (d_{t_\alpha} \leq d_{t_\beta})$. The t-pyramid P_α corresponds to the pyramid that the top t_α characterizes. The functions $u(j)$ ($v(j)$ resp.) indicates the index of the first (the last resp.) t-pyramid to which the job interval i_j belongs. A dominant partial order can now be defined by the the following theorem:

THEOREM 9.1.– *A dominant set of sequences can be constituted by the sequences such that:*

- 1) *tops are sequenced in the ascending order with respect to their index;*
- 2) *only jobs belonging to the first pyramid can be located before the first top and they are sequenced in ascending order with respect to their release dates (or in an arbitrary order in case of release date equality);*
- 3) *only jobs belonging to the last pyramid can be located after the last top and they are sequenced in ascending order with respect to their due dates (or in an arbitrary order in case of release date equality);*
- 4) *only jobs belonging to the t-pyramids P_k or P_{k+1} can be located between two successive tops t_k and t_{k+1} . The jobs belonging only to P_k but not to P_{k+1} are sequenced immediately after t_k according to the ascending order of their due dates (or in an arbitrary order in case of equality). The jobs belonging to both P_k and P_{k+1} are sequenced in an arbitrary order. Lastly to be sequenced are jobs belonging only to P_{k+1} but not to P_k in ascending order with respect to their release dates (or in an arbitrary order in case of equality).*

The fact that theorem 9.1 is relatively insensitive to the variations of the release and due dates of the jobs is another interesting property. Indeed, let us assume that, due to some fluctuations on these dates, the interval structure $\langle I, C \rangle$ becomes $\langle I', C' \rangle$. We note that:

– \preceq_I and $\preceq_{I'}$, the partial orders characterized by theorem 9.1 when it is applied on $< I, C >$ and $< I', C' >$ respectively;

- S_I and $S_{I'}$ the set of tops of $< I, C >$ and $< I', C' >$ respectively;
- $u(i_j)$ and $u(i'_j)$, with $i_j \in I$ and $i'_j \in I'$ respectively, the indexes of the first t-pyramid to which the job interval i_j or i'_j belongs;
- $v(i_j)$ and $v(i'_j)$, with $i_j \in I$ and $i'_j \in I'$ respectively, the indexes of the last t-pyramid to which the job interval i_j or i'_j belongs.

Then we can state that \preceq_I is an extension of $\preceq_{I'}$ if:

$$(S_I = S_{I'}) \wedge (u(i_j) \leq u(i'_j)) \wedge (v(i_j) \geq v(i'_j)) \quad (9.1)$$

In other words, the dominant partial order attached to the problem remains dominant, whatever the variations in the release and due dates of the jobs, as long as condition (9.1) is satisfied. This property is of interest with regard to robustness since we can always be sure that an optimal solution exists in the dominant set of solutions, even when problem parameters are deeply perturbed with respect to condition (9.1).

For instance, let us consider the single machine scheduling problem of Table 9.1 with four jobs. The interval structure attached with such a problem is displayed in Figure 9.6. Two tops $s_1 = 1$ and $s_2 = 4$ can be distinguished which characterize two t-pyramids $P_1 = \{2, 5\}$ and $P_2 = \{2, 3\}$. Let us note that, with respect to definition 9.3, a top does not belong to the pyramid it characterizes.

Jobs	1	2	3	4	5
r_i	6	1	21	24	4
d_i	13	37	33	31	17
p_i	4	5	8	6	7

Table 9.1. A single machine problem $1|r_j|L_{\max}$

When applying theorem 9.1, a dominant set \mathcal{S}_{dom} of cardinality $\text{card}(\mathcal{S}_{\text{dom}}) = (1+1)^2 \cdot (2+1)^1 = 12$ is characterized. It includes the job sequences (see Figure 9.7):

$2 \prec 5 \prec 1 \prec 3 \prec 4, 2 \prec 5 \prec 1 \prec 4 \prec 3, 5 \prec 1 \prec 2 \prec 3 \prec 4, 5 \prec 1 \prec 2 \prec 4 \prec 3,$

$5 \prec 1 \prec 3 \prec 4 \prec 2, 5 \prec 1 \prec 4 \prec 3 \prec 2, 2 \prec 1 \prec 5 \prec 3 \prec 4, 2 \prec 1 \prec 5 \prec 4 \prec 3,$

$1 \prec 5 \prec 2 \prec 3 \prec 4, 1 \prec 5 \prec 2 \prec 4 \prec 3, 1 \prec 5 \prec 3 \prec 4 \prec 2, 1 \prec 5 \prec 4 \prec 3 \prec 2.$

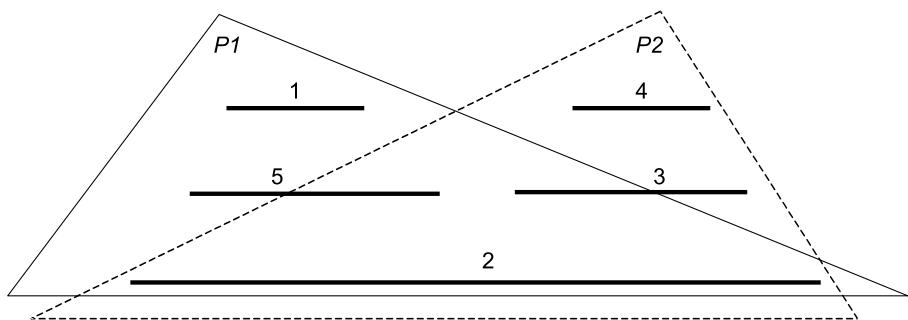


Figure 9.6. Interval structure of the problem described in Table 9.1

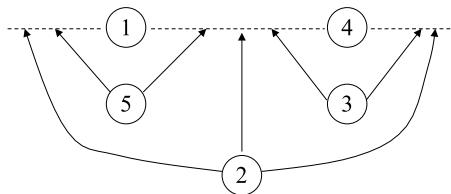


Figure 9.7. Dominant set of job sequences associated with the example

Let us note that, with respect to the dominance property, any job sequence $s \notin \mathcal{S}_{\text{dom}}$ has its minimum L_{\max} greater than or equal to at least one sequence belonging to \mathcal{S}_{dom} . We also highlight that, in the case where the release date of Job 2 is greater than that of Job 1, in such a way that Job 2 would only belong to P_2 , \mathcal{S}_{dom} will remain dominant (see condition (9.1)). Indeed, in this case, the dominant set $\mathcal{S}'_{\text{dom}}$ which corresponds to such a new interval structure would be strictly included in \mathcal{S}_{dom} .

Therefore, the partial order defined by theorem 9.1 characterizes a set of sequences which are potentially better than the other sequences. In order to have a numeric measure of the quality of a dominant set, it is interesting to compute for each job i the best and the worst lateness (denoted as L_i^{\min} and L_i^{\max} respectively), among all the sequences belonging to \mathcal{S}_{dom} . Fortunately, it is possible to compute these values avoiding the complete enumeration of the job sequences belonging to \mathcal{S}_{dom} . Indeed, the computation of L_i^{\min} and L_i^{\max} can be made in $O(n \log n)$, as described in [BRI 07]. We notice that these values depend on r_i , d_i and p_i . Computing the L_i^{\min} and L_i^{\max} of every job can be done in $O(n^2 \log n)$. These values allow us to build up a *lateness diagram*, such as the one displayed in Figure 9.8, with regard to the previous example.

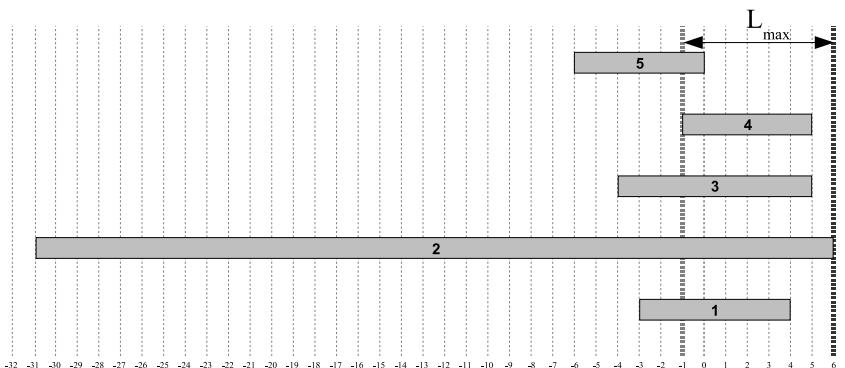


Figure 9.8. A lateness diagram

A major piece of information given by the lateness diagram is an upper-bound and a lower-bound of the criterion L_{\max} : $\max_{i \in T}(L_i^{\min}) \leq L_{\max} \leq \max_{i \in T}(L_i^{\max})$. Such bounds can be used inside a branch-and-bound procedure in order to characterize the set of optimal sequences belonging to \mathcal{S}_{dom} . Such a procedure is proposed in [BRI 07] and avoids the enumeration of the optimal job sequences. It progressively develops a search space where each node corresponds to a specific interval structure. At each step, the branching scheme consists of selecting a pivot job p , corresponding to a top, and another non-top job i . Then two children are generated, considering either the cases where $p \prec i$ or $i \prec p$. These precedence relations can be expressed by means of updating of the interval structure of the parent node, i.e. $r_i \leftarrow r_p$ and $d_i \leftarrow d_p$ respectively. At the end of the procedure, the search tree leaves are such that $\max_{i \in T}(L_i^{\min}) = \max_{i \in T}(L_i^{\max})$. Then, each leaf characterizes a set of optimal job sequences.

A lateness diagram can also be used as a decision-aiding tool aiming at finding a relevant trade-off between flexibility and performance [BRI 03]. In this case, a scheduler is free to select any job x in order to decrease the value of L_x^{\max} which does not satisfy him. Then a procedure, similar to that previously described, can progressively eliminate from \mathcal{S}_{dom} the worst job sequence for x , so decreasing the L_x^{\max} value. At each step, the scheduler obtains knowledge of the remaining number of job sequences, and the current values of L_i^{\min} and L_i^{\max} for each job i , so he becomes able to arbitrate his choices with regard to the performance vs. flexibility trade-off.

9.5.2. The two-machine flow-shop problem

This subsection takes an interest in the famous $F2|prmu|C_{\max}$ problem. It aims at defining a sufficient partial order, which characterizes a large set of optimal sequences,

with regard to the minimization of the maximum completion time C_{\max} . Traditionally, we denote by T the set of n jobs to sequence on two machines M_1 and M_2 (p_{j1} and p_{j2} respectively indicate the processing times of each job $j \in T$ on M_1 and M_2). Only permutation sequences are considered (i.e. $F2|prmu|C_{\max}$) since they are dominant for $Fm ||C_{\max}$, if $m \leq 3$ [CON 67].

For this problem, let us notice first that a well-known sufficient order (further denoted as \preceq_J) has been established by Johnson [JOH 54] in 1954, who stated the sufficient condition of optimality: $\min(p_{i1}, p_{j2}) \leq \min(p_{j1}, p_{i2}) \iff i \preceq_J j$. Following this rule, a single optimal job sequence can be computed in $O(n \log n)$. Nevertheless, because the number of optimal sequences characterized by Johnson's partial condition is relatively low, we present below another partial order which allows us to characterize a larger set, this set necessarily including any Johnson's sequences together with numerous other optimal job sequences. For additional details on this work, the reader should refer to [BRI 06].

Many researchers have taken an interest in enumerating a large number of optimal sequences for $F2|prmu|C_{\max}$. The use of the algorithm proposed in [BEL 82] gives the list of all the sequences satisfying Johnson's rule. Billaut and Lopez [BIL 98] also proposed an algorithm which enumerates, by job permutations inside optimal Johnson's sequences, the complete set of optimal sequences. A related approach based on the notion of *maximum sequence* was also proposed in [BEN 00]. In both the last cases, we notice that only small problems, with a dozen jobs at most, can be solved due to a prohibitive time complexity of the algorithms. On the basis of a given optimal job sequence, Esswein *et al.* also proposed a *greedy forward grouping* algorithm, with worst case time complexity $O(n \log n)$, which determines an optimal group sequence of jobs while maximizing the flexibility (i.e. the number of groups is minimized).

In order to define our partial order, two particular interval structures $\langle I_1, C_1 \rangle$ and $\langle I_2, C_2 \rangle$ are defined:

- I_1 is-the interval structure associated with the jobs $j \in T$ such that $p_{j1} \leq p_{j2}$;
- I_2 is-the interval structure associated with the jobs $j \in J$ such that $p_{j2} \leq p_{j1}$.

An interval $i_j = [p_{j1}, p_{j2}]$ is associated with each job $j \in I_1$ and similarly, an interval $i_j = [p_{j2}, p_{j1}]$ is associated with each job $j \in I_2$. Therefore, a job j such that $p_{j1} = p_{j2}$ belongs to both interval structures $\langle I_1, C_1 \rangle$ and $\langle I_2, C_2 \rangle$ and its corresponding interval is a point. Moreover, we can see that the interval structures $\langle I_1, C_1 \rangle$ and $\langle I_2, C_2 \rangle$ do not change as long as the relative order of the processing times p_{j1} and p_{j2} remains unchanged.

In the following, we focus on the bases of $\langle I_1, C_1 \rangle$ and $\langle I_2, C_2 \rangle$ and on the b-pyramids that they characterize. We assume that the n_1 bases of $\langle I_1, C_1 \rangle$ are indexed in ascending order with respect to their processing times on the first machine (in an arbitrary order in the case of equality). Similarly, the n_2 bases of $\langle I_2, C_2 \rangle$ are indexed, starting from the index $n_1 + 1$, in the descending order with respect to their processing times on the second machine (in an arbitrary order in the case of equality). For each job j , we denote $u(j)$ ($v(j)$ resp.) the index of the base of the first (the last resp.) b-pyramid to which the job j belongs. For a base b_i , we set $u(b_i) = v(b_i) = i$.

For the problem instance of Table 9.2, the interval structures $\langle I_1, C_1 \rangle$ and $\langle I_2, C_2 \rangle$ are represented in Figure 9.9. First, we notice that $\langle I_1, C_1 \rangle$ contains three bases, $b_1 = 1, b_2 = 2$ and $b_3 = 3$, which involve three b-pyramids: $P_1 = \{5, 6\}$, $P_2 = \{6, 7\}$ and $P_3 = \{8\}$. The interval structure $\langle I_2, C_2 \rangle$ only contains a single base, $b_4 = 4$, which involves the b-pyramid $P_4 = \{7, 9\}$. Job 7 is such that $p_{71} = p_{72} = 7$, both belong to $\langle I_1, C_1 \rangle$ and $\langle I_2, C_2 \rangle$.

Jobs	1	2	3	4	5	6	7	8	9
x_i	1	3	8	8	2	4	7	9	5
y_i	6	8	12	2	4	5	7	11	3

Table 9.2. A $F2|prmu|C_{\max}$ problem instance

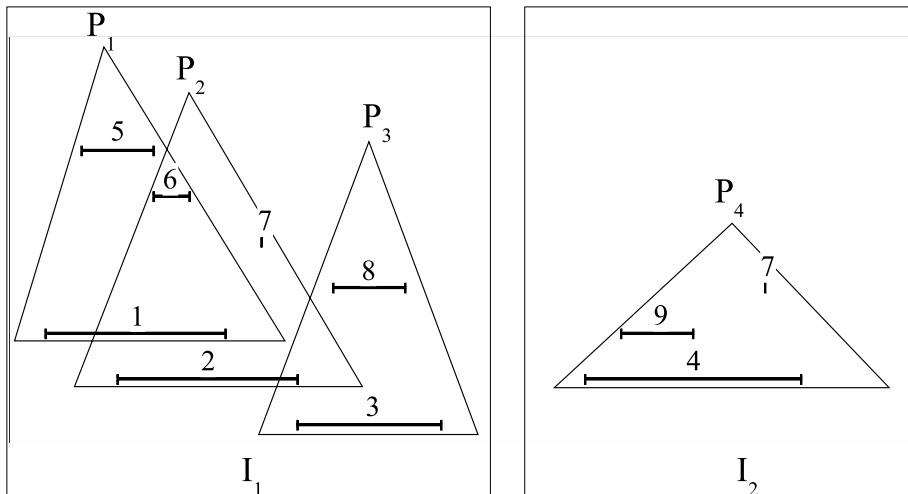


Figure 9.9. Interval structures I_1 and I_2 for the problem of Table 9.2

We denote σ_j , with $j \in [1, n_1]$, the sub-sequence of jobs located between the bases b_j and b_{j+1} . Similarly, we denote σ_k , with $k \in [n_1 + 1, n_1 + n_2]$, the sub-sequence of jobs located between the bases b_{k-1} and b_k . Now, we focus on job sequences in the form: $b_1 \prec \sigma_1 \prec b_2 \prec \sigma_2 \prec \dots \prec b_{n_1} \prec \sigma_{n_1} \prec \sigma_{n_1+1} \prec b_{n_1+1} \prec \dots \prec \sigma_{n_1+n_2} \prec b_{n_1+n_2}$, as illustrated in Figure 9.10.



Figure 9.10. General structure of a job sequence

The theorem below has been stated in [BRI 06]:

THEOREM 9.2.– Any job sequence such that:

- 1) the bases of I_1 and I_2 are sequenced in ascending order with respect to their indexes;
- 2) any job j is sequenced inside any sub-sequence from $\sigma_{u(j)}$ to $\sigma_{v(j)}$ in any order; is optimal.

Theorem 9.2 defines a sufficient partial order which characterizes a large set S_{opt} of optimal job sequences. We recall that this set necessarily includes any Johnson's sequence. Without counting the number of possible permutation inside each subsequence σ_i , the number of optimal sequences can be expressed by the formula: $\text{card}(S_{\text{opt}}) = \prod_{q=1}^N (q+1)^{n_q}$ where n_q corresponds to the number of intervals exactly belonging to q b-pyramids and N is the total number of b-pyramids. In order to illustrate how impressive the number of characterized sequences can be, let us consider a problem with 20 jobs such that $m = 4$, $u(j) = v(j) \forall j \in J$ and $n_i = 4$. In this case, $(4!)^4 = 331776$ optimal sequences are characterized!

Another interest of such an approach lies in the fact that the set S_{opt} is relatively insensitive to processing time fluctuations. In other words, as with the approach described for the single machine problem, even if processing times vary significantly, the set S_{opt} may remain optimal. This is a nice property for designing robust scheduling methods.

Let us turn back to the example of Table 9.2. Theorem 9.2 defines the possible job assignments which are displayed in Figure 9.11. Because the job order inside every subsequence σ_i is free, 13 optimal sequences are characterized (see Figure 9.12) as

having a $C_{\max} = 59$. By comparison, let us note that there are only three Johnson's sequences that are optimal for this example.

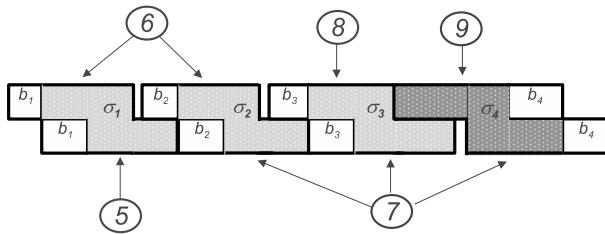


Figure 9.11. Possible job assignments

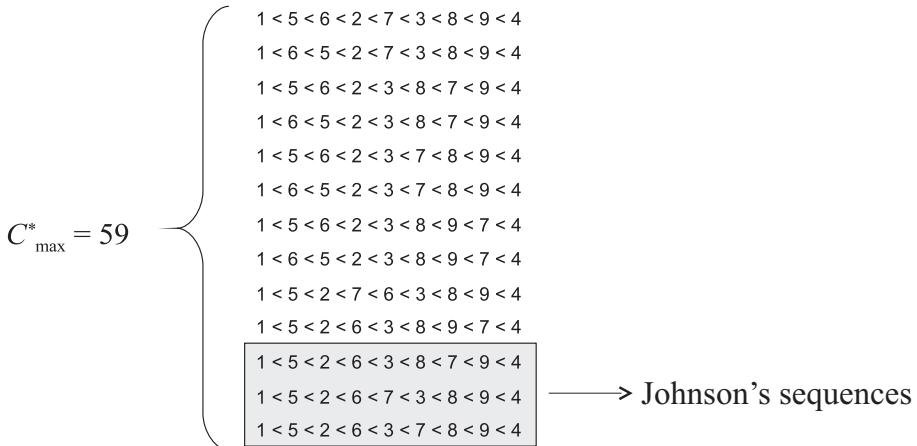


Figure 9.12. Set of characterized optimal sequences

9.5.3. Future prospects

The problems considered in this section, while rather academic, shed light on the advantages that the use of dominance or sufficient conditions can bring when searching for optimal or near-optimal schedules. We also show how interval analysis can be profitable for determining dominance conditions which characterize a large set of dominant solutions having many interesting properties (insensitivity to problem parameter fluctuations, known worst performance, known cardinality). These properties are quite interesting for designing robust scheduling approaches since such approaches often involve the need of a trade-off between performance and flexibility.

Of course, the results presented in this section cannot be directly generalized to more realistic scheduling problems. This is why the determination of more generic dominance conditions, which can be used for instance in flow-shop or job-shop environments, seems a promising research track. For this purpose, as suggested by Cheng *et al.* [CHE 02], more sophisticated hypothesis frames have to be explored, taking, for instance, the relative order of the processing times, the earliest starting times and the latest finishing times of the activities all together into account.

9.6. Conclusion

Constraint-based approaches seem particularly promising with regard to robust scheduling. Indeed, whatever the considered kind of condition (necessary, sufficient or dominance conditions), the analysis of the constraints attached to a problem make it possible to characterize a solution set with a known performance and a known flexibility. These features are of interest, especially when such approaches are used inside a decision-aided process. Moreover, these approaches are relatively insensitive to problem parameter fluctuations, which is another advantage with regard to the robust scheduling field. Indeed, considering approaches based on necessary conditions of feasibility, it is possible to represent the decision variables as intervals of values, each interval value having a uniform probability. Now, considering approaches based on sufficient or dominance conditions such as those detailed in this chapter, the insensitivity to parameter fluctuations is induced by the considered hypothesis frames which only take the relative order of the data into account, and not their exact values. We also highlight the interest of the interval structure notion which has been widely used in this chapter.

We underline that the dominance conditions which have been detailed in this chapter are weakly generic since they are specific to the considered scheduling problems. An interesting approach will be to study dominance conditions in order to enlarge their application frame to more complex problems. Another interesting prospect will be to study how dominance and necessary conditions can be used together in the same approach in order to take benefit from both techniques. For instance, a dominant partial order can be used inside some propagation rules for deducing that a set of variable instantiations is not dominant and can be forbidden, which will cause some adjustments of variable domains.

9.7. Bibliography

- [ALL 81] ALLEN J.F., “An interval based representation of temporal knowledge”, *7th International Joint Conference on A.I. (IJCAI-81)*, Vancouver, Canada, 1981.

- [BAP 01] BAPTISTE P., LE PAPE C. and NUIJTEN W., *Constraint-based Scheduling*, Kluwer Academic Publishers, Boston/Dordrecht/London, 2001.
- [BAP 02] BAPTISTE P., Résultats de complexité et programmation par contraintes pour l'ordonnancement, Habilitation à Diriger des Recherches, Université de Technologie de Compiègne, 2002.
- [BEL 82] BELLMAN R., ESOGBUE A.O. and NABESHIMA I., *Mathematical Aspects of Scheduling and Applications*, Pergamon Press, Oxford, 1982.
- [BEN 00] BENOIT M. and BILLAUT J.-C., “Characterization of the optimal solutions of the $F2||C_{\max}$ scheduling problem”, *2nd Conference on Management and Control of Production and Logistics (MCPL'2000)*, Grenoble, France, 2000.
- [BIL 98] BILLAUT J.-C. and LOPEZ P., “Enumeration of all optimal sequences in the two-machine flowshop”, *2nd IMACS Multiconference on Computational Engineering in Systems Applications (CESA'98)*, Hammamet, Tunisia, p. 378–382, April 1998.
- [BRI 03] BRIAND C., LA H.T. and ERSCHLER J., “Ordonnancement de problèmes à une machine : une aide à la décision pour un compromis flexibilité vs. performance”, *5^e Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF'2003)*, Avignon, France, p. 146–147, 2003.
- [BRI 06] BRIAND C., LA H.T. and ERSCHLER J., “A new sufficient condition of optimality for the two-machine flowshop problem”, *European Journal of Operational Research*, vol. 169, p. 712–722, 2006.
- [BRI 07] BRIAND C., LA H.T. and ERSCHLER J., “A robust approach for the single machine scheduling problem”, *Journal of Scheduling*, vol. 10, p. 209–221, 2007.
- [BRU 94] BRUCKER P., JURISCH B. and SIEVERS B., “A branch and bound algorithm for the job-shop scheduling problem”, *Discrete Applied Mathematics*, vol. 49, no. 1, p. 107–127, 1994.
- [BRU 02] BRUCKER P., “Scheduling and constraint propagation”, *Discrete Applied Mathematics*, vol. 123, p. 227–256, 2002.
- [CAR 75] CARLIER J., Ordonnancement à contraintes disjonctives, Thesis, University of Paris VI, 1975.
- [CAR 89] CARLIER J. and PINSON E., “An algorithm for solving the job-shop problem”, *Management Science*, vol. 35, no. 2, p. 164–176, 1989.
- [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.
- [CAR 00] CARLIER J. and NÉRON E., “A new LP-based lower bound for the cumulative scheduling problem”, *European Journal of Operational Research*, vol. 127, no. 2, p. 363–382, 2000.

- [CAR 04] CARLIER J. and PINSON E., “Jackson pseudo-preemptive schedule and cumulative scheduling problems”, *Discrete Applied Mathematics*, vol. 145, no. 1, p. 80–94, 2004.
- [CAS 94] CASEAU Y. and LABURTHE F., “Improved CLP Scheduling with task intervals”, VAN HENTENRYCK P. (Ed.), *Logic Programming, 11th International Conference on Logic Programming*, Santa Margherita Ligure, Italy, MIT Press, p. 369–383, 1994.
- [CAS 96] CASEAU Y. and LABURTHE F., “Cumulative scheduling with task intervals”, *3rd Joint International Conference and Symposium on Logic Programming*, Bonn, Germany, 1996.
- [CHE 02] CHENG J., STEINER G. and STEPHENSON P., “Fast algorithms to minimize the makespan or maximum lateness in the two-machine flow-shop with release times”, *Journal of Scheduling*, vol. 5, p. 71–92, 2002.
- [CON 67] CONWAY R., MAXWELL W. and MILLER L., *Theory of Scheduling*, Addison Wesley, Reading Mass., 1967.
- [COU 79] COUZINET-MERCÉ C., Études de l’existence de solutions pour certains problèmes d’ordonnancement, Thesis, Paul Sabatier University, 1979.
- [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.
- [DOR 00] DORNDORF U., PESCH E. and PHAN HUY T., “Constraint propagation techniques for the disjunctive scheduling problem”, *Artificial Intelligence*, vol. 122, p. 189–240, 2000.
- [ERS 76] ERSCHLER J., Analyse sous contraintes et aide à la décision pour certains problèmes d’ordonnancement, PhD Thesis, Paul Sabatier University, Toulouse, France, 1976.
- [ERS 82] ERSCHLER J., FONTAN G., MERCÉ C. and ROUBELLAT F., “Applying new dominance concepts to job schedule optimization”, *European Journal of Operational Research*, vol. 11, p. 60–66, 1982.
- [ERS 83] ERSCHLER J., FONTAN G., MERCÉ C. and ROUBELLAT F., “A new dominance concept in scheduling n jobs on a single machine with ready times and due dates”, *Operations Research*, vol. 31, no. 1, p. 114–127, 1983.
- [ESQ 92] ESQUIROL P., HUGUET M.-J. and LOPEZ P., “Time bounds on node graph for scheduling”, *3rd International Workshop on Project Management and Scheduling*, Como, Italy, 1992.
- [ESQ 95] ESQUIROL P., HUGUET M.-J. and LOPEZ P., “Modeling and managing disjunctions in scheduling problems”, *Journal of Intelligent Manufacturing*, vol. 6, p. 133–144, 1995.
- [ESQ 99] ESQUIROL P., LOPEZ P. and MANCÉL C., Représentation et traitement du temps en ordonnancement, Report no. 99455, LAAS-CNRS, Toulouse, October 1999.

- [ESQ 08] ESQUIROL P., LOPEZ P. and HUGUET M.-J., "Constraint propagation and scheduling", in LOPEZ P. and ROUBELLAT F. (Eds.), *Production Scheduling*, ISTE-Wiley, London, 2008.
- [FIS 63] FISHER H. and THOMPSON G.L., "Probabilistic learning combinations of local job-shop scheduling rules", in MUTH J. F. and THOMPSON G.L. (Eds.), *Industrial Scheduling*, p. 225–251, Prentice Hall, Englewood Cliffs, NJ, 1963.
- [FON 80] FONTAN G., Notion de dominance et son application à l'étude de certains problèmes d'ordonnancement, PhD Thesis, Paul Sabatier University, Toulouse, 1980.
- [HER 05] HERROELEN W. and LEUS R., "Project scheduling under uncertainty: survey and research potentials", *European Journal of Operational Research*, vol. 127, no. 2, p. 289–306, 2005.
- [JOH 54] JOHNSON S.M., "Optimal two and three stage production schedules with set-up times included", *Naval Research Logistics Quarterly*, vol. 1, p. 61–68, 1954.
- [LAB 03] LABORIE P., "Algorithms for propagating resource constraints in AI planning and scheduling: existing approaches and new results", *Artificial Intelligence*, vol. 143, no. 2, p. 151–188, 2003.
- [LEV 99] LEVY M.-L., LOPEZ P. and PRADIN B., "Décomposition temporelle et caractérisation de solutions admissibles pour le problème d'ordonnancement à une machine", *RAIRO-Recherche Opérationnelle/Operations Research*, vol. 33, no. 2, p. 185–208, 1999.
- [LOP 91] LOPEZ P., Approche énergétique pour l'ordonnancement de tâches sous contraintes de temps et de ressources, PhD Thesis, Paul Sabatier University, Toulouse, France, 1991.
- [LOP 92] LOPEZ P., ERSCHLER J. and ESQUIROL P., "Ordonnancement de tâches sous contraintes : une approche énergétique", *RAIRO Automatique, Productique, Informatique Industrielle*, vol. 26, p. 453–481, 1992.
- [LOP 98] LOPEZ P., LEVY M.-L. and PRADIN B., "Characterisation by decomposition in scheduling", *Computers in Industry*, vol. 36, no. 1-2, p. 113–116, 1998.
- [LOP 03] LOPEZ P., Approche par contraintes des problèmes d'ordonnancement et d'affectation : structures temporelles et mécanismes de propagation, Habilitation à Diriger des Recherches, Institut National Polytechnique, Toulouse, France, 2003.
- [MAR 96] MARTIN P. and SHMOYS D.B., "A new approach to computing optimal schedules for the job-shop scheduling problem", *5th International Conference on Integer Programming and Combinatorial Optimization, IPCO'96*, Vancouver, British Columbia, p. 389–403, June 1996, LNCS, vol. 1084, Springer Verlag, Cunningham W.H., McCormick S.T. and Queyranne M. (Eds.).

- [MOR 01] MORRIS P., MUSCETTOLA N. and VIDAL T., “Dynamic control of plans with temporal uncertainty”, *17th International Joint Conference on A.I. (IJCAI-01)*, Seattle (WA, USA), 2001.
- [NUI 94] NUIJTEN W.P.M., Time and resource constrained scheduling – a constraint satisfaction approach, PhD Thesis, Eindhoven University of Technology, 1994.
- [PÉR 96] PÉRIDY L., Le problème de job-shop : arbitrages et ajustements, PhD Thesis, University of Technology of Compiègne, 1996.
- [PHA 00] PHAN HUY T., *Constraint Propagation in Flexible Manufacturing*, LNEMS, volume 492, Springer Verlag, Beckmann M. and Künzi H.P. (Eds.), 2000.
- [ROY 70] ROY B., *Algèbre moderne et théorie des graphes*, vol. II, Dunod, Paris, 1970.
- [SCH 98] SCHWALB E. and VILA L., “Temporal constraints: a survey”, *Constraints: An International Journal*, vol. 3, no. 2-3, p. 129–149, 1998.
- [TOR 99] TORRES P., LOPEZ P. and ESQUIROL P., “Lattice of task intervals: a support for edge-finding in disjunctive scheduling”, *4th International Conference on Industrial Engineering and Production Management (IEPM'99)*, Glasgow, Scotland, p. 158–166 (vol. 2), 1999.
- [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, p. 332–343, 2000.
- [TOR 00b] TORRES P. and LOPEZ P., “Overview and possible extensions of shaving techniques for job-shop problems”, *Proc. of the Second International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'00)*, Paderborn, Germany, p. 181–186, March 2000.

Chapter 10

Scheduling Operation Groups: A Multicriteria Approach to Provide Sequential Flexibility

10.1. Introduction

In comparison with pure proactive approaches and pure reactive approaches, robust scheduling literature lacks proactive-reactive scheduling approaches, even if – in general – the latter appear to be more appropriate (see for instance [DAV 00, HER 05] or [LAM 08]).

In addition, among the methods providing some off-line flexibility to scheduling solutions in order to use this flexibility on-line, the majority only considers temporal flexibility which is only the lowest degree of flexibility in scheduling (see section 1.4). In particular, very few studies consider sequential flexibility. One reason for this is the difficulty attached to the design of a method that would produce a set of solutions and allow an evaluation of the worst schedule of this set without enumerating its elements. As well as approaches described in Chapters 9 and 11 (see also [ALO 02, BRI 06, BRI 07, ART 07]), the concept of *groups of permutable operations* depicted in this chapter has this particularity.

We present in this chapter two robust-scheduling proactive-reactive approaches. They are based on the concept of *groups of permutable operations*. The first method

Chapter written by Carl ESSWEIN, Jean-Charles BILLAUT and Christian ARTIGUES.

described in section 10.3 is called ORABAID¹. Section 10.4 describes a modification of the proactive part of ORABAID called AMORFE² and initially proposed in [ESS 03a]. It has been applied to several scheduling problems and a summary of these applications is given in section 10.5.

10.2. Groups of permutable operations

10.2.1. History, principles and definitions

The notion of *groups of permutable operations* was born 30 years ago at the CNRS “LAAS” laboratory in Toulouse (France). It was proposed to represent a particular class of schedule sets, within the development framework of a decision-aid approach for real-time shop scheduling. Even if it was not explicitly qualified this way, this method was nothing but a robust scheduling approach. Called ORABAID, it is implemented in the software ORDO^R ([ROU 95]) used by many French companies and has the following two goals:

- 1) to propose, at every decision point, a set of actions compatible with the constraints (including due date constraints if possible),
- 2) to update this set of actions, taking into account all the actual events and decisions taken in real-time in the shop.

To reach these goals, the choice has been made to propose, instead of a single reference schedule, a set of reference schedules, and then to use this set to have several alternatives at every decision point. Obviously, any of the choices given should lead to the execution of a schedule that is *feasible*, that is, that satisfies all the hard constraints of the model, and *acceptable*, that is, that respects the due date as long as it is possible. In other words, this approach is a proactive-reactive approach whose proactive “module” characterizes a set of schedules (defines the set but does not enumerate its elements) that are feasible and acceptable regarding the decision-maker preferences.

Since this characterization cannot efficiently define the whole set of feasible and acceptable schedules (see [ERS 76]), the development of ORABAID turns, beginning with the work of Demmou in [DEM 77], to the search for a subset of admissible schedules by means of sufficient conditions of admissibility. This gave birth to the

1. ORABAID is the French acronym for “Ordonnancement d’Atelier Basé sur l’Aide à la Décision”, that is, “shop scheduling based on decision aid”.

2. AMORFE is the French acronym for “Approche Multicritère pour l’Ordonnancement Flexible”, that is, “multicriteria approach for flexible scheduling”.

concept permutable operation groups: the definition of such sufficient conditions is done by the definition of a sequence of groups of permutable operations on each resource.

DEFINITION 10.1.– *A group of permutable operations is a set of operations consecutively processed on the same machine, in an order that is not fixed in advance.*

In what follows, as long as there is no ambiguity, a group of permutable operations is simply called a *group*. A *group schedule* is then defined by a sequence of groups on each machine. A group schedule is called *feasible* if any permutation of operations inside each group leads to a schedule that satisfies the problem constraints. As a consequence, any feasible group schedule characterizes, without our needing to enumerate them, a whole set of feasible schedules, defined by all the combinations of permutable operations inside each group.

DEFINITION 10.2.– *The quality of a group schedule is the quality of the worst semi-active schedule it characterizes.*

This choice has been made for several reasons, the most important being that considering the worst case, which allows us to provide a *performance guarantee* for the group schedule.

A great advantage of the use of group schedules is that this particular class of set of schedules is such that “*the degrees of freedom, related to the processing sequences on the machines, are brought to light in a particularly pleasant way*” [THO 80].

Through many studies (including [ERS 76, DEM 77, THO 80, LEG 89, BIL 93, ART 97]), the ORABAID approach has been extended step by step. It is now able to take into account a large set of constraint types. For instance, it can handle scheduling problems with both disjunctive and cumulative resources, non-linear routes, generalized precedence constraints, multi-resource operations, setup-times etc.

In the rest of this chapter, for ease of reading, we consider a simplified scheduling problem, that is, with only disjunctive resources, linear routes and no multi-resource operation. Unless specified differently we consider the traditional job-shop scheduling problem as defined in [BRU 07]. The ORABAID and AMORFE approaches described in sections 10.3 and 10.4 can be extended to much more general problems.

In order to illustrate these definitions, let us consider the 3-machine job-shop instance defined in Table 10.1. j refers to the job and i refers to the position of the

j	1			2			3		
i	1	2	3	1	2	3	1	2	3
$M_{j,i}$	1	2	3	2	3	1	3	1	2
$p_{j,i}$	3	3	3	4	3	1	2	2	2

Table 10.1. A three-machine job-shop instance

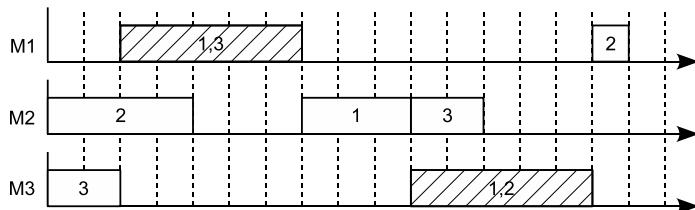
operation in the route. Figure 10.1(a) represents a feasible group schedule for this instance. Only the job numbers are given since we can easily check the routes. This group schedule is made up of 7 groups: two two-operation groups and five groups with a single operation.

As long as we only consider semi-active schedules, choosing a particular sequence of operations inside each group leads to one of the four schedules of Figure 10.1(b). The group schedule has a C_{\max} of 17 which is given by the fourth depicted characterized schedule.

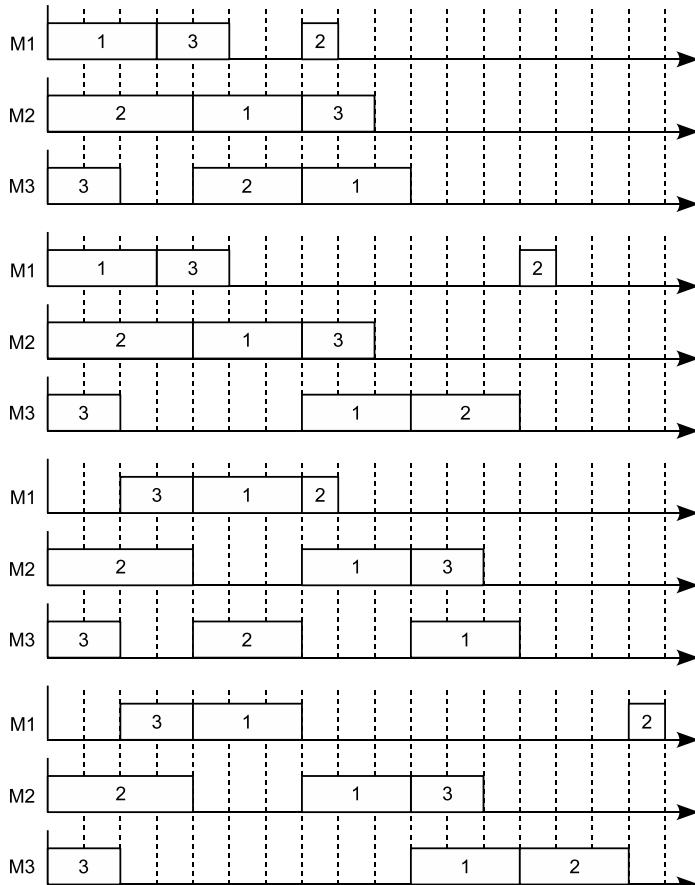
10.2.2. Representation and evaluation

To use the concept of group schedule, it is necessary to find a way to evaluate such a structure. More precisely, given a group schedule, we should be able to guarantee it only characterizes schedules whose quality is over a given threshold. In brief, the quality of the worst characterized schedule should be efficiently evaluable. The set of semi-active schedules that a group schedule implicitly defines is such that this evaluation is computable in polynomial time. In our opinion, this is another great advantage of representing a set of schedules by means of a group schedule.

To evaluate this “worst-case quality”, each operation $O_{j,i}$ is associated with a *worst-case latest completion time*, denoted by $\gamma_{j,i}$. This completion time $\gamma_{j,i}$ is such that, whatever the sequencing choices made in each groups, it is guaranteed that $C_{j,i} \leq \gamma_{j,i}$, assuming – of course – that we only consider semi-active schedules. The computation of the different $\gamma_{j,i}$ requires the computation of all the *worst-case earliest start times*, denoted by $\mu_{j,i}$. In what follows, as long as there is no ambiguity, we omit “worst-case” when talking about worst-case earliest start time and/or worst-case latest completion times.



(a) A group schedule...



(b)... and the semi-active schedules it characterizes.

Figure 10.1. A feasible group schedule characterizes several feasible semi-active schedules

10.2.2.1. Earliest start time computation

The earliest start-time $\mu_{j,i}$ of operation $O_{j,i}$ is defined as the smallest start time compatible with the whole set of permutable operations within each group. We can easily notice that only the permutations inside groups including operations *preceding* $O_{j,i}$ and linked to $O_{j,i}$ by a constraint, are involved in the computation of $\mu_{j,i}$. The constraints implicated are of two kinds: the routes and those related to disjunctive use of resources. It follows that:

- $O_{j,i}$ cannot start before the completion of operation $O_{j,i-1}$ which is inside a group that may contain other operations,
- $O_{j,i}$ cannot start before the completion of all the operations inside the group processed just before the group of $O_{j,i}$ on its resource.

This simple remark enables the identification of the set of constraints involved in the computation of $\mu_{j,i}$.

For the first case, operation $O_{j,i-1}$ is placed in the worst position regarding the consequences on operation $O_{j,i}$. This worst case happens when $O_{j,i-1}$ is processed in the last position in his group and when the first operation of this group is the one with the greatest earliest start time. We then have two sub-cases depending on whether or not $O_{j,i-1}$ has the greatest earliest start time of the group. These sub-cases are illustrated in Figure 10.2 where $G(j, i - 1)$ denotes the operation group $O_{j,i-1}$.

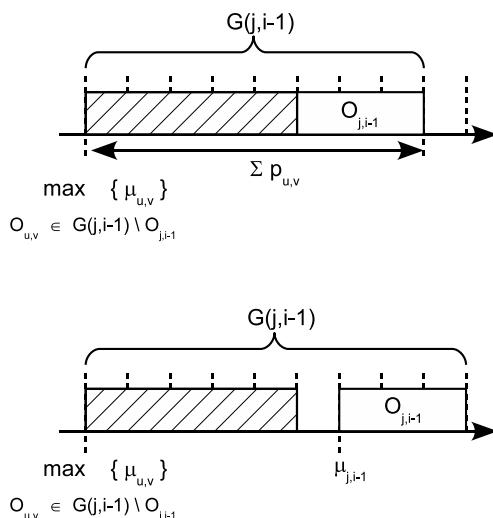


Figure 10.2. Worst case routing constraints

Thus, the routing constraint can be expressed in the following way:

$$\forall(j, i) \in [1, \dots, n] \times [2, \dots, m],$$

$$(A) \quad \mu_{j,i} \geq \max_{O_{u,v} \in G(j,i-1) \setminus O_{j,i-1}} \{\mu_{u,v}\} + \sum_{O_{u,v} \in G(j,i-1)} p_{u,v}$$

$$(B) \quad \text{and } \mu_{j,i} \geq \mu_{j,i-1} + p_{j,i-1}$$

The second sub-case (line (B)) does not consider group considerations: it is the simple expression of classical *routing constraints* in (no-group) scheduling. Conversely, the first sub-case involves all the operations in the group including operation $O_{j,i-1}$. The related constraints are referred to as *group constraints*.

Let us now consider *resource constraints*. We use $M_{j,i}$ to denote the machine used to process operation $O_{j,i}$, and $G_k^{M_{j,i}}$ the group in k th position on machine $M_{j,i}$. The fact that $O_{j,i}$ could not be processed until all the operations in the group preceding $G(j, i)$ are finished can be formalized in the following way:

$$\forall(j, i) \in [1, \dots, n] \times [1, \dots, m], \quad \text{with } G(j, i) = G_k^{M_{j,i}},$$

$$\mu_{j,i} \geq \max_{O_{u,v} \in G_{k-1}^{M_{j,i}}} \{\mu_{u,v}\} + \sum_{O_{u,v} \in G_{k-1}^{M_{j,i}}} p_{u,v}$$

In the particular case of the the first operation group on a resource, the resource constraints are simply given by:

$$\mu_{j,i} \geq 0.$$

To make the computations easier, the choice has been made to represent a group schedule by means of an operation-on-node graph ([THO 80]). This representation is not given here but the interested reader could find more details about it in [THO 80, LEG 89] or [BIL 93]. Let us just indicate that operations (including a few dummy operations) are represented by nodes and that each type of constraint is represented by a set of arcs. Computing a longest path in this graph gives the earliest start time of all the operations and the earliest completion time of all the jobs in polynomial time.

Another graph definition, presented in [ESQ 95] and applied to groups in [ART 05], allows this computation. It has the advantage of also computing the

worst-case earliest completion times of operations. It is worth noticing that defining the worst-case earliest completion time of an operation from the worst-case start time is not obvious due to the fact that the worst-case for the completion time of an operation involves all the permutation of the operations inside the group it belongs to, which is not the case for the worst-case start time. So the authors propose some kind of generalization of the previous graph representation which is done by defining two nodes for each operation as well as some arcs to model the three sets of constraints.

10.2.2.2. Latest completion time computation

Similarly, such an enumeration of constraint types allows the computation of the latest completion times. However, to give a sense to the qualifier “latest”, we should previously define upper bounds for the completion time of jobs. We consider that each job has a due date d_j . As defined in the introduction, a schedule is called feasible if it respects all constraints except the due date. A schedule is acceptable if it is feasible and, in addition, satisfies the due dates. By extension a schedule set is feasible if all its represented schedules are feasible and it is admissible if all its represented schedules are admissible. The idea behind the definition of the upper bound limit of the job completion times is to refuse to increase lateness of jobs already late, and to allow non-late jobs to complete at their due date. Thus, for non-late jobs the upper bound is fixed to their due date, when for the late jobs it is fixed to the earliest completion time found previously.

Symmetrically with what is done when computing the earliest start time, the only operations involved in computing the latest completion time $\gamma_{j,i}$ are those following $O_{j,i}$. Yet, the method proposed in [THO 80], [LEG 89] and [BIL 93] leads to the computation of the latest completion time of each job, denoted by γ_j for any j , in polynomial time.

10.2.2.3. Quality of a group schedule

So given a group schedule, the previous technique allows us to compute in polynomial time the worst case completion time of every job, that is, the worst value that could take each C_j from among the set of characterized schedules, whatever the size of this set. It is important to notice that this evaluation is tight and that the technique also makes it possible to determine the schedule with this value of C_j . As a consequence, the following result about the evaluation of any regular *minmax* criteria Z can be announced. The proof can be found in [ESS 03a].

THEOREM 10.1.– Given a group schedule OG and a regular criteria $Z = \max_{j=1,\dots,n} f_j(C_j)$, the exact evaluation of the worst schedule characterized by OG can be performed in polynomial time.

So, even if the number of characterized schedules is arbitrarily high, the exact evaluation of the group schedule remains efficient.

Unfortunately, this nice property does not hold for *minsum* criteria. Indeed, for any *minsum* criteria $Z = \sum_{j=1,\dots,n} f_j(C_j)$, we only obtain an upper bound of the evaluation since the worst case for job $k1$ may be different from the worst case for job $k2$.

A proactive-reactive approach based on group scheduling still remains of interest for *minsum* criteria. We now simply require a group schedule evaluation method. Interested researchers may then either develop a dedicated method or take one from the literature. For instance, the evaluation techniques developed by Aloulou and Portmann (see Chapter 11) and dealing with criteria maximization problems could be applied here, since a group schedule is nothing but a particular partial order between operations.

10.3. The ORABAID approach

10.3.1. *The proactive phase: searching for a feasible and acceptable group schedule*

The first of the two phases of a robust scheduling approach based on group schedules consists of building off-line a group schedule that will be on-line the reference solution for the second phase. The method developed for building such a group schedule in the ORABAID approach is split into three consecutive steps:

- search for a *feasible* group schedule,
- attempt to make this schedule *acceptable*,
- attempt to increase its flexibility.

First, some groups are built and sequenced on the resources by means of a simple greedy procedure. Then, if the group schedule is not acceptable, two techniques are combined and applied iteratively to shorten the shortest paths associated with late jobs: changing the resource allocation of a critical operation, and splitting a critical group into two consecutive groups. Should there remain late jobs after having applied each of these techniques, the due date of the latest job would be artificially reduced to its earliest computed completion time and the procedure iterated once again; that is, until

the group schedule is completely acceptable (with some potential modified due dates). Finally, the third step consists of making some merges of consecutive groups without violating admissibility or acceptability constraints. This last step aims at increasing the number of operations per group and thus the sequential flexibility available for the reactive phase.

The detailed description of this three-step approach can be found in [THO 80]. It has then been successively extended to more general problems in [LEG 89, BIL 93, ART 97]. We present here its main ideas.

10.3.1.1. *Construction of a feasible group schedule*

Except for the resource assignment problem which we omit here to ease presentation, two issues have been generally treated, in terms of obtaining a feasible group schedule: partition the operations on each resource inside groups, and sequence the groups on each resource. These two connected problems are solved together by means of a simple greedy procedure:

At each iteration of the procedure:

- 1) The set of candidate operations is defined as the set of operations whose predecessors are already scheduled.
- 2) A resource is selected among the resources where candidate operations are assigned to. In order to favor active schedule generation³, the selected resource is the one that will advance the candidate operation completing first if selected.
- 3) Concerning the use of this resource, we have to select one operation among candidate operations to be assigned to it. This is done by means of a priority rule trying to respect due dates as much as possible: slacks of candidate operations are computed and the selected operation is the one with the lowest *slack over remaining processing time* ratio (SLACK/RPT rule).
- 4) Finally, if this does not violate the feasibility constraints of the partial group sequence, the selected operation is added to the preceding group on the resource it is assigned to.

3. Because of permutability conditions, the construction of active schedules is neither ensured nor necessarily desirable in the strict sense.

10.3.1.2. *Searching for acceptability of the group schedule*

Once a feasible group schedule is obtained from the previous step, its acceptability for the decision-maker is not guaranteed: some jobs may be late in at least one characterized schedule. The aim of the second step is to reduce the length of the critical paths associated with these late jobs.

The proposed procedure identifies critical groups and iteratively realizes two modifications on them: modifications of operations assignments, and group splits. The first category consists of calling into question the decision of assigning an operation to a resource (impossible in the job-shop case) and assigning it to another, trying to insert it in another existing group. The second kind consists of splitting a critical group into two consecutive groups still processed on the same resource. Doing this ensures that the set of characterized schedules is modified and the interested reader can find in [LEG 89] proofs that the detailed rules used ensure that the obtained group schedule is closer to acceptability. These two techniques are applied successively while acceptability is not reached while there remain untried modifications.

10.3.1.3. *Increasing the group schedule flexibility*

In the third and final step of the proactive phase of the method described above we try to decrease the total number of groups and to provide the maximum of flexibility for the reactive phase. This is done by merging two consecutive groups. Some rules can be defined (see [THO 80]) to establish whether merging two consecutive groups can be done without modifying the feasibility of the group schedule.

At the end of the three steps that constitute the proactive phase of the method, a group schedule is obtained. The procedure is such that it builds a group schedule that is feasible (all the characterized schedules satisfy the problem constraints) and tends to be acceptable (due dates are respected as much as possible).

10.3.2. *The reactive phase: real-time decision aid*

The reactive phase of the global approach corresponds to the actual execution of the group schedule that has been obtained. It is managed by using an interactive decision support system (described in [BIL 93, BIL 96] or [ART 08]) that works as follows:

- the state of the workshop is characterized by the state of the resources and of the operations. This state is regularly updated and is assumed to be perfectly known,
- the state modifications are due to events and decisions,

– decision making is based on the group schedule – which is updated according to the decisions – and on the decision support system.

The decision aid is based on the definition of the *sequential slack*. The sequential slack of operation $O_{j,i}$, denoted by $ss(j,i)$, is equal to the minimum between the *proper slack* and the *group slack* defined as follows:

$$ss(j,i) = \min \left(\gamma_{j,i} - p_{j,i} - \mu_{j,i}; \min_{O_{u,v} \in G(j,i) \setminus O_{j,i}} \gamma_{u,v} - \sum_{O_{u,v} \in G(j,i)} p_{u,v} - \mu_{j,i} \right)$$

The sequential slack of an operation corresponds to the maximum increase of its starting time without modifying the group schedule feasibility. It constitutes a good measure for the feasibility of a group schedule. The sequential slacks are permanently updated for all the operations that have not been started.

To decide which operation of a group should be started, the decision aid uses these indicators as follows: the best choice of an operation inside a group is the one with the maximum sequential slack [THO 80]. This decision allows us to increase the slacks of the other operations of the group, which leads to a better situation.

Of course, the decision maker can take another decision. He has the ability to simulate a decision and seeing the consequences of this decision. The decision maker can then validate this decision or not.

10.3.3. Some conclusions about ORABAID

The interest of this method lies in the introduction of the concept of permutable operations associated with an efficient method for evaluating the worst schedule, and in the definition of the sequential slack, making it possible to preserve flexibility and feasibility in real time. The value of the approach is demonstrated by its integration in the ORDO^R software widely used by French companies.

However, concerning the evaluation of the best characterized solution or about the quality control of the worst characterized solution and of the flexibility proposed to the decision maker, this approach only give partial answers. The objective of the AMORFE method is to give a more satisfactory answer to these questions.

10.4. AMORFE, a multicriteria approach

In this section, we only consider the off-line problem of constructing a group sequence. We explain why this problem can be approached from a multicriteria perspective and we propose a new resolution method called AMORFE.

10.4.1. Flexibility evaluation of a group schedule

The slacks defined in ORABAID (sequential slack, proper slack, group slack) measure the temporal flexibility associated with the operations. Here we propose to give an evaluation of the sequential flexibility and in the following, flexibility will implicitly concern sequential flexibility.

The flexibility of a group schedule is related to the total number of groups, denoted by $\#Gps$. Indeed, if the number of groups is small, it means that operations are gathered and thus that the number of characterized sequences is high and that the solution is more flexible. Thus, maximizing the flexibility is closely related to minimizing the number of groups.

As an illustration, let us consider the example of Figure 10.3 for a two-machine job-shop problem. The first group schedule contains eight groups and characterizes four semi-active schedules. The second schedule contains four groups and characterizes 144 semi-active schedules.

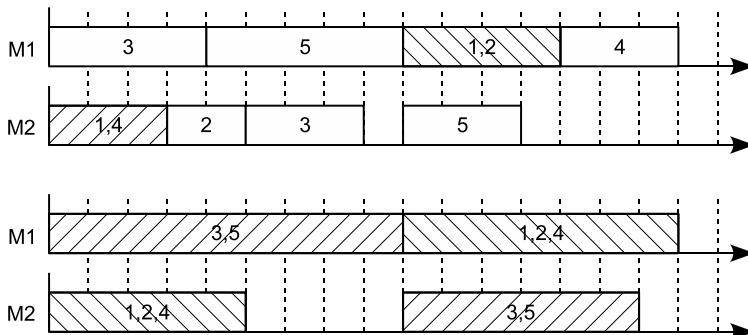


Figure 10.3. The number of groups making it possible to determine the sequential flexibility of a group sequence

It is clear that the number of groups is not systematically related to the number of characterized sequences. For instance, two groups each containing n operations characterize less semi-active schedules than one group with one operation plus a second group with $2n - 1$ operations. Thus, the number of groups is an approximation of the number of characterized sequences, but this objective has some merits: the measure $\#Gps$ can take a polynomial number of different values (at most one group per operation), which is not the case for the number of characterized semi-active schedules.

We propose minimizing the number of groups in order to maximizing the flexibility of a group schedule, denoted by ϕ . The relation between the two measures is the following:

$$\phi = \frac{NbOp - \#Gps}{NbOp - m}$$

where $NbOp$ is the total number of operations to schedule and m the number of machines.

ϕ represents the “grouping rate” of the operations in the group schedule: the less flexible case corresponds to the case where each group contains exactly one operation (no flexibility). In this case, $\#Gps = NbOp$ and thus $\phi = 0$. Conversely, when there is one group per machine – which is possible for a flow-shop for instance – we have $\#Gps = m$ and then $\phi = 100\%$.

In the example of Figure 10.3, the first group schedule has a flexibility equal to $\phi = 2/8 = 25\%$, whereas the flexibility of the second group schedule is equal to $\phi = 6/8 = 75\%$.

Note that in general for the job-shop problem, the minimum number of groups is strictly greater than m , because of the routing constraints making one group per machine impossible. Thus, it is generally impossible to obtain a flexibility of 100%. Of course, this comment is invalid for flow-shop or open shop configurations.

10.4.2. Evaluation of the quality of a group schedule

Remember the definition of the quality of a group schedule (see definition 10.2). The characterization of a solution set by a group schedule leads to the problem of defining the quality of this solution set. We denote by Z the cost of minimizing one schedule. The group schedule, denoted by GS , is characterized by a finite set of values: at most the number of semi-active characterized schedules (several schedules may have the same value of criterion Z). Because there are no hypotheses about uncertainties and because they are not modeled by random variables, the mean value is not considered for the evaluation process of the group schedule. However, two other significant values can be used: the smallest – which corresponds to the best characterized solution – and the biggest – which corresponds to the worst characterized solution – denoted by Z^{best} and Z^{worst} . We say that Z^{best} is the quality of the group schedule in the best case and Z^{worst} to the quality in the worst case.

The quality in the worst case is the one that is optimized by the ORABAID approach. This makes it possible to provide a solution set for which a performance is guaranteed and a lot of users say that this approach is very useful for the decision maker who prefers a set of good solutions instead of the best but unique and non-robust solution. We agree with this remark and this is the reason why the quality of the group sequence is defined by the quality in the worst case.

However, we believe that two other measures have to be taken into account. First, we believe that the optimization process has to consider the flexibility of the proposed solution. Second, we believe that considering the quality in the best case during the construction of a group schedule allows more information to be given to the decision maker, thereby improving the proposed solution.

10.4.3. Some considerations about the objective function definition

If the three measures were not conflicting, optimizing one will also optimize the others. However, flexibility and quality are in the worst case conflicting measures, as illustrated in Figure 10.4, where it is assumed that $Z = C_{\max}$.

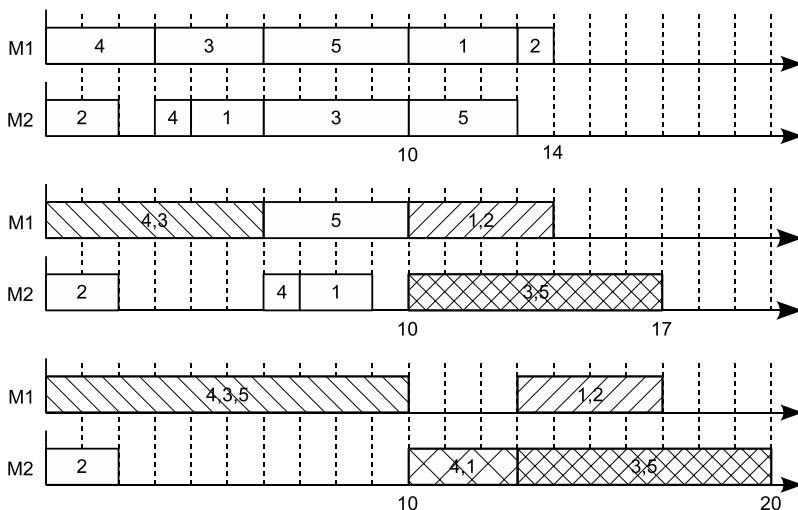


Figure 10.4. Conflicting aspects of flexibility and quality in the worst case

Three group schedules are represented in Figure 10.4. The first is a schedule of operations without groups that can be considered as a group schedule with $\#Gps = 10$. Its flexibility is equal to 0% and the worst C_{\max} (the only one) is equal to 14. The second group schedule contains $\#Gps = 7$ groups. Its flexibility is

$\phi = 3/8 = 37.5\%$ and its worst case quality is equal to $C_{\max} = 17$. The last group schedule contains $\#Gps = 5$ groups. Its flexibility is equal to $\phi = 62.5\%$ and the worst possible makespan is equal to 20. We easily understand that the greater the flexibility, the greater the number of characterized schedules and the worst the quality of the worst schedule.

Thus, we have to consider at least two conflicting criteria, ϕ and Z^{worst} , which justifies a multicriteria approach. We propose to use the ϵ -constraint approach. Remember that with this approach, all the criteria except one are bounded and the last is optimized. We have chosen this approach for several reasons related to the properties of the approach. We refer to [T'k 06] for a detailed description of the approach. It allows a clear definition of the objective function (better than a linear combination of criteria) and the method can easily be implemented in an interactive approach [STE 86]. At each step of such an interactive approach, the decision-maker can modify the bounds on the criteria and drive the solving method in the direction of its preferred compromise solution.

We will now present some algorithms that could be used as the basics of such an interactive method.

We consider three criteria to optimize: Z^{best} , Z^{worst} and $\#Gps$ and each time the resolution method is used in the interactive method, one criterion has to be minimized, whereas the two other criteria are bounded. The role of criterion Z^{best} is not crucial since this criterion is not in conflict with Z^{worst} . This is the reason why we consider in the following the minimization of $\#Gps$ subject to a bound on Z^{worst} and the minimization of Z^{worst} subject to a bound on $\#Gps$.

It is easier for the decision maker to indicate the worst case quality he can accept rather than the flexibility, so we consider in the following the problem of minimizing the number of groups subject to the respect of a bound on the quality of the worst characterized schedule. Such a problem is illustrated in Figure 10.5.

In the following, if (P) denotes a scheduling problem, (P_G) will denote the same problem as (P) with the additional constraint that we build group sequences instead of operation sequences. The objective function is denoted by $\epsilon(\#Gps/Z)$, which corresponds to the minimization of $\#Gps$ subject to a bound for the worst value of Z among the characterized sequences. In the three-field notation (see Chapter 1) we use gps for the constraint to build groups of permutable operations. If problem P is denoted by $\alpha|\beta|\gamma$, problem (P_G) is denoted by $\alpha|\beta,gps|\epsilon(\#Gps/\gamma)$.

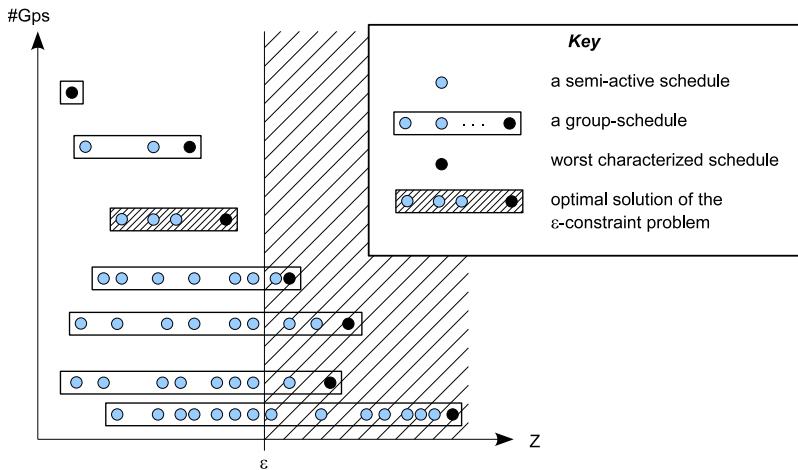


Figure 10.5. ϵ -constraint approach: maximization of the flexibility subject to a given guarantee for the quality in the worst case

Z^{best} may also play an important role during the construction of a group sequence. The way of considering this criterion is discussed in the next section.

10.4.4. Quality guarantee in the best case

10.4.4.1. Advantages

The quality evaluation of a set of semi-active schedules in the worst case is a difficult problem if the objective function is arbitrary. However, for any regular criterion of type *minmax*, it is possible with a group sequence to evaluate exactly and in polynomial time the worst characterized solution. However, this is not possible for the best characterized solution. Indeed, such an evaluation is equivalent to minimizing the objective function, with the additional constraints imposed by the group sequences, which is an NP-hard problem. Thus, determining Z^{best} is generally impossible in polynomial time.

A solution is to build a group schedule for which the quality in the best case is greater than a given bound Z_0 . For this, we do not need to know the best characterized schedule: having a sufficiently good schedule may be sufficient.

10.4.4.2. Respect for quality in the best case

In order to respect the quality in the best case of a group schedule, we propose to solve the group scheduling problem *with fixed sequences*. The problem is defined as follows.

Let (P) be a scheduling problem and (P_G) the corresponding group scheduling problem, as defined in section 10.4.3.

DEFINITION 10.3.– Let us consider problem (P) where the sequences on each machine are fixed. We denote by σ_i the sequence on machine M_i , $i = 1, \dots, m$. The group scheduling problem with fixed sequences associated with problem (P) , is the problem (P_G) with the additional constraint to characterize the sequences σ_i , $\forall i = 1, \dots, m$.

To ensure that the sequences σ_i are characterized by the group schedule, two operations $O_{j,i}$ and $O_{u,v}$ can be assigned to two successive groups G_k^ℓ and G_{k+1}^ℓ only if $O_{j,i}$ precedes $O_{u,v}$ in the fixed sequence corresponding to the machine. In other words, two arbitrary operations performed by a machine M_ℓ can be assigned to the same group only if all the operations between them in sequence σ_ℓ are also assigned to the same group.

The value of this restriction is that it provides a guarantee in the best case. Indeed, suppose we have a good schedule s , solution of the scheduling problem without group. The group scheduling problem with fixed sequences defined by schedule s insure that the best characterized solution will be at least as good as solution s , because s belongs to the set of characterized sequences.

Thus, we propose to build group schedules by considering fixed sequences, where the sequences are given by solutions of the scheduling problem without group consideration. The AMORFE approach proposed in [ESS 03a] for the determination of a group schedule can be decomposed into two phases:

- 1) compute a solution to problem (P) , without group consideration,
- 2) group the operations as much as possible, so that the initial solution belongs to the set of characterized solutions (*fixed sequences*) and so that a given bound on the worst characterized solution is respected.

This procedure allows us to build a group sequence that is as flexible as possible, that respects a bound on the worst characterized schedule, and that contains a known (good) sequence, i.e. that respects a given quality in the best case.

10.5. Application to several scheduling problems

The AMORFE method has been applied to some traditional scheduling problems: several single machine problems, two-machine flow-shop, job-shop and open shop

problems and the more general m -machine job-shop scheduling problem, with the makespan criterion. We now give some results for the two-machine problems and for the job-shop problem.

In a study concerning two-machine shop scheduling problems (see [ESS 05]), group schedules with both a good flexibility and a good quality can be obtained in polynomial time, even if these problems are NP-hard. The open shop problem is the simpler problem to solve since it is always possible to find a group schedule with optimal quality and at most three groups per machine. This allows us to build group schedules with a huge flexibility. For the two-machine job-shop problem, some particular cases can be solved in linear computation time and the general case study shows that the most difficult case corresponds to the two-machine flow-shop case. However, even if this problem is strongly NP-hard, a huge amount of flexibility can be obtained together with an optimal quality. In other words, for the two-machine flow-shop, a huge number of optimal solutions can be characterized by using group schedules. As an illustration, for instances with 100 jobs and processing times randomly generated between 1 and 100, an algorithm with complexity $O(n \log n)$ can generate group schedules with an average flexibility of 94.8%, which corresponds to the characterization of more than 5×10^{167} optimal schedules.

The results obtained by the AMORFE method applied to the job-shop problem are less extreme. However, some experiments have been made on classical benchmark instances. The results show that it is possible to provide some flexibility together with a high quality and that more flexibility can be obtained with a slight reduction of the quality. “Squared” instances (for which the number of machines equals the number of jobs) are the most difficult ones. For these instances, the flexibility provided without reduction of the quality is small (8 to 15%) whereas for non-squared instances the flexibility comes to 50 to 60%.

The interested reader can find in [ART 05, ESS 03b, ESS 03a] more details concerning the algorithms that implement the AMORFE method, as well as the associated computational results.

To conclude this section, remember that for each problem mentioned before, the proposed algorithms are only one possible way to apply the AMORFE method. Indeed, this method defines a general framework and it is possible to improve the performances obtained for these problems, and particularly for the m -machine job-shop problem.

10.6. Conclusion

We have presented in this chapter two proactive-reactive methods for robust scheduling based on the use of sequential flexibility. The first method that used group schedules is the ORABAID method. The AMORFE method has recently been developed and proposes a modification in the proactive part of ORABAID. This new proactive phase uses a new measure of flexibility and improves the definition of group schedule quality. In the AMORFE method, the notions of flexibility and quality are integrated in a multicriteria approach where the flexibility is maximized subject to a bound for the quality. This approach has the advantage of guaranteeing the quality of the best and the worst characterized schedule, by using the solution of a classical scheduling problem.

Thus, a clear advantage of this approach, from our point of view, is that this proactive-reactive method for solving the scheduling problem under uncertainties, uses traditional scheduling resolution algorithms. In other words, this approach is able to integrate the most recent resolution algorithms – as well as future algorithm discoveries – that have been tested for scheduling problems without uncertainties.

10.7. Bibliography

- [ALO 02] ALOULOU M.A., Structure flexible d'ordonnancements à performances contrôlées pour le pilotage d'atelier en présence de perturbations, PhD Thesis, Institut National Polytechnique de Lorraine, 2002.
- [ART 97] ARTIGUES C., Ordonnancement en temps réel d'ateliers avec temps de préparation des ressources, PhD Thesis, LAAS/CNRS, Paul Sabatier University, Toulouse, France, 1997.
- [ART 05] ARTIGUES C., BILLAUT J.-C. and ESSWEIN C., “Maximisation of solution flexibility for robust shop scheduling”, *European Journal of Operational Research*, vol. 165, no. 2, p. 314–328, 2005.
- [ART 07] ARTIGUES C. and ALOULOU M., “Worst-case evaluation of flexible solutions in disjunctive scheduling problems”, in Gervasi O. and Gavrilova M.L. (Eds.), *Computational Science and Its Applications - ICCSA 2007*, Springer, Lecture Notes in Computer Science 4707 edition, 2007.
- [ART 08] ARTIGUES C. and ROUBELLAT F., “Real-time workshop scheduling”, in Lopez P. and Roubellat F. (Eds.), *Production Scheduling*, ISTE-Wiley, London, 2008.
- [BIL 93] BILLAUT J.-C., Prise en compte des ressources multiples et des temps de préparation dans les problèmes d'ordonnancement en temps réel, PhD Thesis, LAAS/CNRS, Paul Sabatier University, Toulouse, France, 1993.

- [BIL 96] 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.
- [BRI 06] BRIAND C., LA H.T. and ERSCHLER J., “A new sufficient condition of optimality for the two-machine flowshop problem”, *European Journal of Operational Research*, vol. 169, no. 3, p. 712–722, 2006.
- [BRI 07] BRIAND C., LA H.T. and ERSCHLER J., “A robust approach for the single machine scheduling problem”, *Journal of Scheduling*, vol. 10, no. 3, p. 209–221, 2007.
- [BRU 07] BRUCKER P., *Scheduling Algorithms*, Springer Verlag, Berlin, 5th edition, 2007.
- [DAV 00] DAVENPORT A.J. and BECK J.C., “A survey of techniques for scheduling with uncertainty”, available at <http://www.eil.utoronto.ca/profiles/chris/chris.papers.html>, 2000.
- [DEM 77] DEMMOU R., Etude de familles remarquables d’ordonnancements en vue d’une aide à la décision, PhD Thesis, Paul Sabatier University, Toulouse, France, 1977.
- [ERS 76] ERSCHLER J., Analyse sous contraintes et aide à la décision pour certains problèmes d’ordonnancement, PhD Thesis, LAAS/CNRS, Toulouse, France, 1976.
- [ESQ 95] ESQUIROL P., HUGUET M.-J. and LOPEZ P., “Modeling and managing disjunctions in scheduling problems”, *Journal of Intelligent Manufacturing*, vol. 6, p. 133–144, 1995.
- [ESS 03a] ESSWEIN C., Un apport de flexibilité séquentielle pour l’ordonnancement robuste, PhD Thesis, Université François Rabelais, Tours, France, 2003.
- [ESS 03b] ESSWEIN C., PURET A., MOREIRA J. and BILLAUT J.-C., “An efficient method for job shop scheduling with sequential flexibility”, *Second Operational Research Peripatetic Postgraduate Programme (ORP3 2003)*, Lambrecht, Germany, p. 171–181, September 2003.
- [ESS 05] ESSWEIN C., BILLAUT J.-C. and STRUSEVICH V.A., “Two-machine shop scheduling: compromise between flexibility and makespan value”, *European Journal of Operational Research*, vol. 167, no. 3, p. 796–809, 2005.
- [HER 05] HERROELEN W. and LEUS R., “Project scheduling under uncertainty. Survey and research potentials”, *European Journal of Operational Research*, vol. 165, no. 2, p. 289–306, 2005.
- [LAM 08] LAMBRECHTS O., DEMEULEMEESTER E. and HERROELEN W., “Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities”, *Journal of Scheduling*, vol. 11, no. 2, 2008.
- [LEG 89] LEGALL A., Un système interactif d’aide à la décision pour l’ordonnancement et le pilotage en temps réel d’atelier, PhD Thesis, LAAS/CNRS, Paul Sabatier University, Toulouse, France, 1989.

- [ROU 95] ROUBELLAT F., BILLAUT J.-C. and VILLAUMIÉ M., “Ordonnancement d’ateliers : d’Orabaid à Ordo”, *Revue d’automatique et de productique appliquées*, vol. 8, no. 5, p. 683–713, 1995.
- [STE 86] STEUER R.E., *Multiple Criteria Optimization: Theory, Computation, and Application*, John Wiley & Sons, New York, USA, 1986.
- [THO 80] THOMAS V., Aide à la Décision pour l’Ordonnancement d’Atelier en Temps Réel, PhD Thesis, Paul Sabatier University, Toulouse, France, 1980.
- [T’k 06] T’KINDT V. and BILLAUT J.-C., *Multicriteria Scheduling: Theory, Models and Algorithms*, Springer-Verlag, Heidelberg, 2nd edition, 2006.

Chapter 11

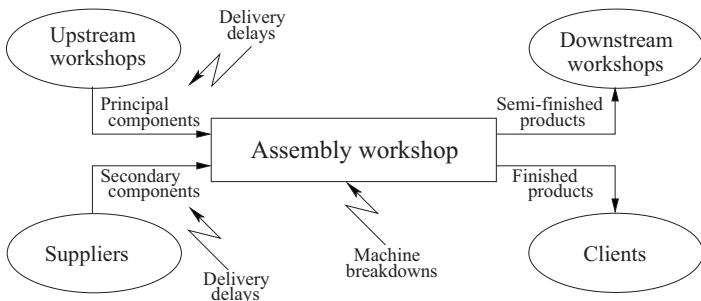
A Flexible Proactive-Reactive Approach: The Case of an Assembly Workshop

11.1. Context

In this chapter, we focus on logistic chain problems encountered in the European Growth Project V-chain GRD1-2000-25881. Our concern is for workshop level interactions between downstream and upstream workshops from the same company and with external suppliers, customers and partners in the organization (see Figure 11.1). In this workshop, semi-finished and finished products are produced and then delivered to downstream workshops and to external customers. A penalty is associated with each delay in relation to delivery dates desired by the internal and external customers. Several components are needed to make the products: main components, delivered by upstream workshops, and intermediate components, purchased from suppliers. After acquisition, intermediate components are stored in the workshop until they get used. We presume that there is a cost proportional to the time these components remain in inventory.

Our objective is to propose a scheduling approach which considers main component delivery dates as communicated by upstream workshops and semi-finished and finished product delivery dates desired by downstream workshops and by external customers. Based on job start dates given by the computed schedule,

Chapter written by Mohamed Ali ALOULOU and Marie-Claude PORTMANN.

**Figure 11.1.** Context of the study

procurement dates of intermediate components are planned and new due dates can be negotiated with downstream workshops and customers to ensure a safety margin. A failure to respect these new dates is also penalized.

Besides the possible existence of disruptions in the assembly workshop (machine failure, increases in job processing time, etc.), the workshop's heavy dependence on its partners increases the potential of uncertainty with arrival dates of main and intermediate components. This makes workshop scheduling especially tricky since activities of the downstream workshops may be planned according to calculated scheduling. Consequently, it is important to plan for flexibility in the solutions proposed to the workshop in order to increase system robustness.

In this chapter, we consider that the assembly workshop is only made up of a single machine. We present a proactive-reactive approach for workshop control in the presence of disruptions liable to occur in the shop or with partners. The key point in this approach is to consider uncertainty in a proactive way (off-line). This is done by the introduction of flexibility in job sequencing and flexibility in time in the solution calculated by the proactive algorithm. In real time, a reactive algorithm knowingly uses the flexibility introduced and proposes sequencing actions to control the workshop. Note that the storage cost of intermediate components is not used to implement our proactive-reactive approach; it is used instead to compare it to a traditional predictive-reactive approach.

The global process that we propose for designing our control model has two phases: definition and implementation. In the model definition phase, we define the scheduling problem involved and its environment, what we call the solution to this problem, as well as the different concepts used to measure the quality of a solution.

In the implementation phase, we design and implement two algorithms: a proactive algorithm and a reactive algorithm.

11.2. Definition of the control model

11.2.1. *Definition of the problem and its environment*

The problem is to schedule on one machine a set of jobs $N = \{T_1, \dots, T_n\}$ (representing finished and semi-finished products). Each job $T_j \in N$ is characterized by a processing time $p_j > 0$, an earliest start date $r_j \geq 0$ (procurement date of main components), a desired due date $d_j \geq 0$ (desired by downstream workshops and customers) and a weight $w_j > 0$ indicating its relative importance. Imperative precedence constraints can exist between jobs. The workshop is subject to disruptions caused by machine failures and to late arrivals of main and/or intermediate components. The disruption characteristics, such as their frequency and time, are not known before execution. Preemption of a job is only authorized if the machine breaks down. In this case, we consider that this job must be restarted after the failure without any modification to its characteristics.

We consider two objective functions to minimize:

- the weighted sum of absolute tardiness $\bar{T}_w = \sum_{1 \leq j \leq n} w_j \max\{0, C_j - d_j\}$, where C_j and $\max\{0, C_j - d_j\}$ are the completion time and absolute tardiness of job j respectively,
- the makespan $C_{\max} = \max_{1 \leq j \leq n} C_j$.

11.2.2. *Definition of a solution to the problem*

As with the approaches proposed in Chapters 9 and 10, a solution to the problem is not a specific schedule but a set of schedules. The main difference is the structure representing a solution to the problem. Esswein *et al.* define a solution by an ordered group assignment defining for each machine a sequence of groups where the operations within a group are totally permutable. Any semi-active schedule satisfying the constraints imposed by the sequence of groups is presumed eligible or feasible (see section 10.2 in Chapter 10 for the definition of feasibility). We propose two new characteristics to define a solution to the problem involved. Our goal is to generate solutions with more flexibility or better performance *a priori* (and, we hope, *a posteriori*) w.r.t. to the solutions given by Esswein *et al.* In order to do this, a solution given by our approach is characterized by a structure defined by the type of

schedules considered: semi-active, active or non-delay (see section 1.1.3 in Chapter 1 for definitions), and by a partial order between jobs. Contrary to Esswein *et al.* and with the objective of obtaining better performances in the worst case *a priori*, we will not only work with semi-active schedules, but we can constrain the space of flexible solutions to active and even non-delay schedules. The partial order must fulfill the constraints of the problem. It is chosen for offering a compromise between flexibility and performance. Note that any ordered group assignment is a partial order between operations, whereas the opposite is not true.

Generally speaking, proposing several high performance schedules for a workshop is more interesting than proposing just one. In fact, the decision-maker can choose among proposed schedules a schedule which will best satisfy his preferences or which responds the best to non-modeled constraints. It is even more interesting if certain proposed schedules have common characteristics making it possible to switch from one schedule to another easily. In this case, the decider can delay the choice of which schedule to execute. Only in real time will he have to gradually make this decision considering the state of the workshop.

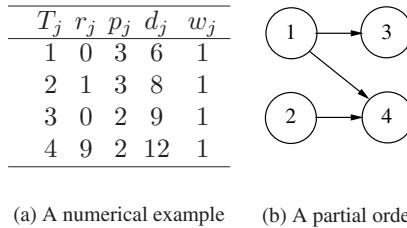
The structure defined by the pair (partial order of jobs, type of schedules) offers several schedules with common precedence properties. In order to go from one schedule to another or from one subset of schedules to another subset, we only need to arbitrate one of several disjunctive arcs so as not to create cycles in the resulting graph and to satisfy the constraints imposed by the type of schedules. A partial order is a natural structure which integrates well in constructive resolution methods. In these methods, each time one or more arcs are added, the transitive graph is retained for easy prevention of cycle formation.

11.2.3. Definition of the solution quality

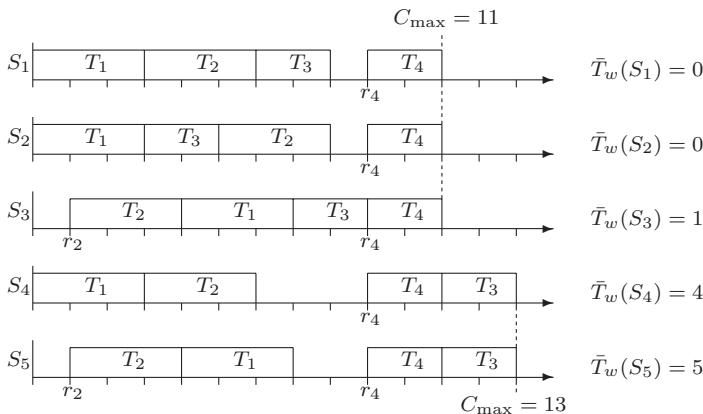
11.2.3.1. Preliminary example

In Figure 11.2, we consider a scheduling problem with four jobs and a partial order in which the only restrictions are that job T_1 precedes jobs T_3 and T_4 and that job T_2 precedes T_4 . This partial order represents two non-delay schedules S_1 and S_2 , three active schedules S_1 , S_2 and S_3 , all of which are of course semi-active (see Figure 11.3). Criteria values are also given in the same diagram.

This example shows that *a priori* quality of a partial order depends on the type of schedules chosen. In fact, considering semi-active schedules offers more represented



(a) A numerical example (b) A partial order

Figure 11.2. A four-job problem**Figure 11.3.** Non-delay, active and semi-active represented schedules

schedules than with other types of schedules thus improving flexibility of the solution. However, the worst performance of represented schedules is not as good as for active and non-delay schedules (see performances of schedules S_4 and S_5).

For the remainder of this chapter, we consider semi-active scheduling and we define the indicators used to measure the *a priori* quality of a solution. Results involving active and non-delay schedules can be found [ALO 07].

11.2.3.2. Performance of a solution

11.2.3.2.1. Definition of performance measures

A priori performance of a solution S depends on the performance of o_S schedules that it represents. In order for S to be chosen, all o_S schedules must have acceptable performance in relation to the optimal performance obtained by solving the original problem without partial order added for criterion $C_{\max}(C_{\max}^*)$ and criterion $\bar{T}_w(\bar{T}_w^*)$.

Since the number of schedules o_S represented can be quite large, it is natural to only consider the most representative of these schedules. To evaluate solution S , we prefer to confine ourselves to schedules providing the best and worst performances for both criteria C_{\max} and \bar{T}_w . The best and worst performances are lower and upper bounds over criteria values following the partial order. They are noted as bC_{\max} for the best and wC_{\max} for the worst makespan, and $b\bar{T}_w$ for the best and $w\bar{T}_w$ for the worst weighted sum of tardiness.

We consider that a solution S is preferred if values $bC_{\max}(S)$ and $wC_{\max}(S)$ (resp. $b\bar{T}_w(S)$ and $w\bar{T}_w(S)$) are close to C_{\max}^* (resp. to \bar{T}_w^*). The following measure $D(S)$ is used to aggregate the four measures:

$$D(S) = \alpha D^1(S) + (1 - \alpha) D^2(S), \quad \alpha \in [0, 1] \quad (11.1)$$

$$D^1(S) = \beta \frac{bC_{\max}(S) - C_{\max}^*}{C_{\max}^*} + (1 - \beta) \frac{wC_{\max}(S) - C_{\max}^*}{C_{\max}^*}, \quad \beta \in [0, 1] \quad (11.2)$$

$$D^2(S) = \gamma \frac{b\bar{T}_w(S) - \bar{T}_w^*}{\bar{T}_w^* + 1} + (1 - \gamma) \frac{w\bar{T}_w(S) - \bar{T}_w^*}{\bar{T}_w^* + 1}, \quad \gamma \in [0, 1] \quad (11.3)$$

S is all the more preferable as $D(S)$ is small.

11.2.3.2.2. Calculation of performance measures

Given a solution S , we must be able to calculate the four values $bC_{\max}(S)$, $wC_{\max}(S)$, $b\bar{T}_w(S)$ and $w\bar{T}_w(S)$ by solving the corresponding minimization and maximization problems. Because of [LAW 73], we know that the minimization problem of makespan, noted as $1|prec, r_j|C_{\max}$, can be resolved in $O(n^2)$ times. The minimization problem of the weighted sum of tardiness, noted as $1|prec, r_j|\bar{T}_w$, is NP-hard in the strong sense [LAW 93]. For this problem, we implemented a genetic algorithm and compared it to priority rule-based heuristics such as ATC, X-RM and KZRM [MOR 93]. Results have shown that the genetic algorithm gives better results than the other algorithms but requires more calculation time to converge [ALO 01].

To calculate the worst makespan and worst weighted sum of tardiness, we have introduced in [ALO 04] new optimization problems. These problems are noted as $1(sa)|prec, r_j|(F \rightarrow \max)$, where F is the objective function to maximize and the sa notation means that semi-active schedules are considered. We showed that problem $1(sa)|prec, r_j|(C_{\max} \rightarrow \max)$ can be resolved in $O(n^2)$ times and that problem $1(sa)|prec, r_j|(\bar{T}_w \rightarrow \max)$ is NP-hard in the strong sense. We developed different

heuristics based on genetic algorithms and priority rule-based heuristics. We obtained the same conclusions as for minimization problems [ALO 01].

In conclusion, given a solution, we use two polynomial time algorithms to calculate the exact values of the best and worst total time and heuristics to calculate approximate values of the best and worst weighted sum of tardiness.

11.2.3.3. *Flexibility of a solution*

The solutions that we propose *a priori* offer two types of flexibility, also called static flexibility: flexibility over sequences and flexibility over time. In what follows, we present the indicators used to measure the static flexibility of a solution.

11.2.3.3.1. Flexibility over sequences

Having different schedules enables the decision maker to have several alternatives for controlling the workshop in real time. Each time the decider must place a job, he could, if the solution allows it, select the job which most aptly fulfills his preferences or non-modeled constraints. This flexibility can also absorb disruptions caused by a delay in delivery of main or intermediate components for example. Consequently, the number of schedules can measure the flexibility over the sequences of the solution. However, the calculation problem of this number is #P-complete [BRI 91]. We propose another measure of flexibility Flex_{seq} given by the number of directionless arcs in the transitive graph representing the partial order. This number is an upper bound of the number of additional sequencing alternatives, which are available only to the decision-maker. The higher this number, the more flexible the solution will be. If we note as U all the arcs in the transitive graph representing the partial order and $|U|$ as the cardinality of this group, we have

$$\text{Flex}_{\text{seq}} = \frac{n(n - 1)}{2} - |U| \quad (11.4)$$

In the proactive algorithm, to limit the number of flexibility classes, we use a transformation of the number of directed arcs in a multi-level qualitative measure. Each level $N_l, l = 1, \dots, nbLevel$, is characterized by an interval $[nbArcsMin_l, nbArcsMax_l]$ representing minimum and maximum values of directed arcs that a solution can have belonging to this level.

11.2.3.3.2. Flexibility over time

Flexibility over sequences generally implies flexibility over time, which is in fact a zero degree of flexibility [GOT 02]. This flexibility is available according to time

intervals in which the jobs can be executed. In the proactive algorithm, we use a global measure of flexibility over time. This measure is given by the ratio between total job margin, for solution S considered, and the total processing time P . It is given by the following equation:

$$\text{Flex}_{\text{time}} = \frac{wC_{\max}(S) - P}{P}. \quad (11.5)$$

11.3. Proactive algorithm

The objective here is to design a proactive algorithm that will calculate solutions with good quality *a priori*. To measure the *a priori* quality of a solution, we use performance and flexibility indicators presented in section 11.2.3. The performance of a solution is linked to the best and worst values of the scheduling criteria that it represents. Flexibility measures are associated with a solution: a flexibility measure over sequences and a flexibility measure over time. The number of quality measures, the NP-completeness of optimization problems associated with some of these measures, and the size of problems involved, led us to not consider an exact method approach. We have focused directly on the use of metaheuristics and more particularly on genetic algorithms.

In what follows, we will present the general diagram of the global algorithm. We will then explain in detail the different steps of this algorithm.

11.3.1. General schema of the proposed genetic algorithm

The algorithm that we propose is intended for the generation of a series of solutions offering a good compromise between several quality measures *a priori*. The objective of the algorithm is to search for one (or more) solution(s) S minimizing the measure of performance $D(S)$, maximizing the flexibility measure over sequences Flex_{seq} and maximizing the measure of flexibility over time $\text{Flex}_{\text{time}}$. The problem considered is then a multicriteria problem. To solve it, the proposed algorithm considers in an iterative way the sub-spaces of solutions defined by the different levels of flexibility over sequences $N_l, l = 1, \dots, nbLevel$ (see section 11.2.3.3). In each sub-space of solutions, the algorithm searches for solutions that minimize a *Fitness* measure, the linear combination between $D(S)$ and $\text{Flex}_{\text{time}}(S)$, given by the following equation.

$$\text{Fitness}(S) = \theta D(S) - (1 - \theta)\text{Flex}_{\text{time}}(S), \theta \in [0, 1]. \quad (11.6)$$

```

For  $l = 1$  to  $nbLevel$  Do
    Generate an initial population  $P_0$  of solutions belonging to level of flexibility
    over sequences  $N_l$ ;
    Evaluate the chromosomes of  $P_0$  and initialize  $Elite_{N_l, \theta}$ ;
    For  $i = 0$  to  $nbGen$  Do
        Select, from  $P_i$ ,  $N_{pop}$  couples of chromosomes for reproduction;
        Crossover selected couples of chromosomes;
        Evaluate generated children and update  $Elite_{N_l, \theta}$ ;
        Mutate with low probability  $\pi_{mut}$  generated children;
        Evaluate mutated children and update  $Elite_{N_l, \theta}$ ;
        Select, from  $P_i$  and generated children,  $N_{pop}$  chromosomes for survival;
    EndFor
EndFor

```

Figure 11.4. Genetic algorithm for set value θ

By varying θ between 0 and 1, we obtain solutions S offering a good compromise between the flexibility measure over time $\text{Flex}_{\text{time}}(S)$ and the performance measure $D(S)$ for a given level of flexibility over sequences N_l . This algorithm is described in Figure 11.4. In this algorithm, we use the following notations:

- $nbGen$ is the number of generations of the single criterion genetic algorithm;
- P_i is the population of iteration solutions i , $i = 0, \dots, nbGen$;
- N_{pop} is the size of a population of individuals;
- $nbLevel$ is the number of flexibility levels over sequences;
- π_{mut} is the mutation probability;
- $Elite_{N_l, \theta}$ is the elite population with a dimension of $sizeElite$ that we obtain at the end of the algorithm for a value of $\theta \in [0, 1]$ and a level of flexibility N_l , $l = 1, \dots, nbLevel$.

After execution, the algorithm returns $Elite_{N_l, \theta}$ sets. The decision maker will be able to select the solution deemed adequate based on the performances and flexibilities that it presents. He will eventually be able to develop and calculate other flexibility indicators not used during the optimization phase or test, in the presence of disruptions, the solutions with simulations.

In practice, we do not have to explore all the research space defined by the flexibility measure over sequences. In fact, the decision maker may:

- either impose a minimum performance limit (corresponding to maximum values of criteria not to exceed) that the solutions must ensure. In this case, we can operate in a dichotomic way (on flexibility levels) and search for solutions belonging to the most

permissive flexibility level possible while guaranteeing the restrictions in performance criteria,

- or request the search for the most powerful solutions belonging to two or three given flexibility levels for example.

11.3.2. Selection and strategy of reproduction

At each iteration of the genetic algorithm, we use two selection procedures. The first selects N_{pop} couples of chromosomes for reproduction. The second selects, among the children generated and the old population, N_{pop} chromosomes which will survive and form the new population. In our implementation, we use the roulette technique [GOL 89]. N_{popIn} is the number of different chromosomes in the current population. In this population, we select N_{popOut} chromosomes to reproduce as pairs or to survive in the next generation. A new force equal to N_{popIn} acts upon the best individual, while the second chromosome receives a force equal to $N_{popIn} - 1$ and the last one experiences a force equal to 1. The chromosomes are selected with a probability that is proportional to their force.

11.3.3. Coding of a solution

To code a partial order, between the jobs to execute, characterizing a solution S , we use a square ternary matrix MT with a dimension of $n \times n$, n being the number of jobs to execute [DJE 96, POR 98]. Matrix $MT = (MT(i, j))_{1 \leq i, j \leq n}$ is defined by

$$MT(i, j) = \begin{cases} 1 & \text{if } T_i \text{ precedes } T_j \text{ in } S, \\ -1 & \text{if } T_j \text{ precedes } T_i \text{ in } S, \\ 0 & \text{if } T_i = T_j \text{ or } T_i \text{ and } T_j \text{ are permutable in } S. \end{cases} \quad (11.7)$$

The resulting matrix is transitive and anti-symmetric. During implementation, we can only memorize the higher diagonal matrix ($\frac{n(n-1)}{2}$ elements instead of n^2 elements). Nevertheless, for a better understanding, the extended matrix is used here.

MT coding is a direct coding because there is bijective correspondence between the space of partial sequences and the space of ternary matrices.

11.3.4. Crossover operator

The crossover operator introduces two chromosomes or parents to generate one, two or more chromosomes or children. The crossover operator is the most important operator in genetic algorithms and it must enable the efficient use of research space.

The crossover is made between two individuals selected by a procedure which favors the strongest individuals. During the generation of children, it is advisable to retain some of the important properties of parents. We propose a crossover inspired by the MT3 crossover proposed by [DJE 96]. It guarantees the retention of precedence constraints common to both parents, including imperative precedence constraints.

Figure 11.5 presents a version of the crossover algorithm for generating a child, represented by a ternary matrix F , from two parents represented by matrices $MT1$ for the father and $MT2$ for the mother respectively. This algorithm is not given in detail for the sake of simplicity.

-
- Step 1. $F \leftarrow \frac{MT^1 + MT^2}{2}$;
Update $nbArcs(F)$;
- Step 2. Modify one or two values $F(i, j)$ to ensure that the generated child is different from both parents;
Update $nbArcs(F)$;
- Step 3. While $nbArcs(F) < nbArcs(MT^1)$ Do
 Randomly select one of the parents (parent 1 with probability π and parent 2 with probability $1 - \pi$); let MT be the corresponding matrix;
 Select two jobs i and j s.t. $F(i, j) = 0$ and $MT(i, j) \neq 0$;
 $F(i, j) \leftarrow MT(i, j)$;
 $F(j, i) \leftarrow -F(i, j)$;
 Compute the transitive closure of F and update $nbArcs(F)$;
End While
- Step 4. If $nbArcs(F) \leq nbArcsMax$ then F is retained;
else F is discarded;
-

Figure 11.5. Modified MT3 crossover

11.3.5. Mutation operator

The mutation operator is used to guarantee the diversity of the chromosome population. The mutation we propose consists of changing the order of at least two jobs. It is described in Figure 11.6, where MT and F represent the matrix of the solution to mutate and the matrix of the mutant respectively.

Step 1. $F \leftarrow F_0$; /* F_0 is the matrix representing imperative precedence constraints*/

Step 2. Randomly select two jobs i and j s.t. $MT(i, j) \neq 0$;

$F(i, j) \leftarrow -MT(i, j)$;

$F(j, i) \leftarrow -F(i, j)$;

While $nbArcs(F) < nbArcs(MT)$

 Randomly select two jobs i and j s.t. $F(i, j) = 0$ and $MT(i, j) \neq 0$;

$F(i, j) \leftarrow MT(i, j)$;

$F(j, i) \leftarrow -F(i, j)$;

 Compute the transitive closure of F ;

End While

Step 4. If $nbArcs(F) \leq nbArcsMax$ then F is retained;

else F is discarded;

Figure 11.6. Mutation MUT3

11.4. Reactive algorithm

11.4.1. Functions of the reactive algorithm

The reactive algorithm ensures workshop control. It has three main functions. The first one is to control the execution of jobs by following the partial order characterizing the retained solution, and by respecting constraints caused by the type of scheduling chosen. The second function is to react to disruptions. The third function is to detect when the current solution no longer guarantees the decision maker's expectations.

The reactive algorithm is based on the partial order defining the solution chosen in the proactive phase. The question is how:

- each time a sequencing decision must be made, to offer different alternatives respecting the partial sequence, thus increasing the potential of being able to absorb late arrivals of raw material,

- to preserve flexibility over time until a breakdown occurs,

- to obtain good performance if no disruption occurs during the execution of jobs.

It is generally impossible to fulfill all these objectives simultaneously. For example, in order to retain as much flexibility over sequences as possible, the algorithm may choose to leave the machine idle for a longer period of time. This leads to a loss of flexibility over time and can result in bad performance at the end of the execution.

In what follows, we proceed in two phases. In the first phase, we presume that there is no disruption. We propose control algorithms and indicators for measuring their capacity to fulfill the three objectives discussed. In the second phase, we consider that disruptions can occur during job execution. At the start, we use the same procedures as in the first step. When a disruption occurs, an analysis module is used to find out if the partial order will absorb it or if modifications must be made to the partial order to limit performance loss. When performance loss is high, intermediate actions must be proposed while waiting for the construction of a new proactive solution for the execution of the remaining jobs.

11.4.2. Reactive algorithms in the absence of disruptions

We will first present indicators which measure the aptitude of a reactive algorithm to fulfill the three objectives discussed in the previous section. These indicators are called *a posteriori* quality measures. We will then propose different control algorithms.

11.4.2.1. *A posteriori* quality measures

11.4.2.1.1. *A posteriori* flexibility

A posteriori flexibility measures the capacity of the reactive algorithm to use flexibility in the sequences of a solution. Consider the solution, given in Figure 11.7, of a 5 job scheduling problem. This solution represents 15 semi-active schedules. Its flexibility over sequences equals $\text{Flex}_{\text{seq}} = 6$, since the number of directed arcs in the transitive graph representing the partial sequence is 4 ($= 10 - 6$).

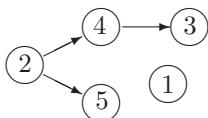
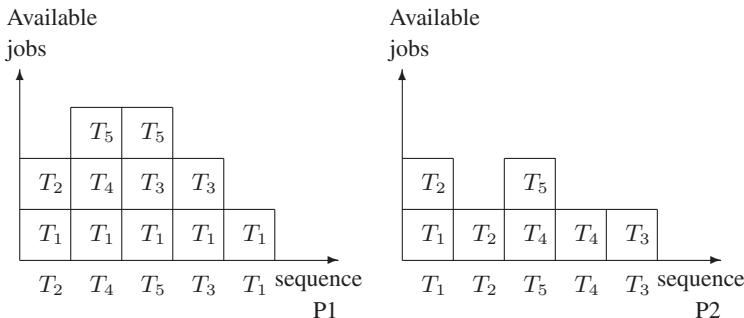


Figure 11.7. A solution to a 5 job problem

In Figure 11.8, we represent two schedules calculated by two reactive algorithms P_1 and P_2 respecting the partial order in Figure 11.7. The available jobs at each iteration are represented in the vertical axis. At first, jobs T_1 and T_2 are available. The P_1 algorithm chooses to schedule T_2 first. In the second phase, jobs T_1 , T_4 and T_5 are available. P_1 chooses to schedule T_4 . The P_1 algorithm is applied until all jobs are executed. The resulting sequence is presented in the horizontal axis; it is sequence $(T_2, T_4, T_5, T_3, T_1)$ for P_1 .

**Figure 11.8.** Sequences given by $P1$ and $P2$

We observe that $P1$ offers two sequencing alternatives in the first phase (jobs $T1$ and $T2$), three alternatives in the second and third phases, two alternatives in the fourth phase and one alternative in the last phase. However, $P2$ only offers two alternatives in the first and third phases and one alternative in the other phases. In summary $P1$ proposes a total of $6 + 5$ alternatives to the decider whereas $P2$ only offers $2 + 5$ alternatives. Based on this example, we propose a measure that provides the proportion of flexibility over sequences of a solution S used by an algorithm P . This measure is computed by the following formula

$$F_{\text{flex}}(S, P) = \frac{\text{Number of alternatives proposed by } P - n}{\text{Flex}_{\text{seq}}(S)} \quad (11.8)$$

If $F_{\text{flex}}(S, P)$ is very close to 100%, then P is able to use flexibility over sequences contained in S . For our example, we have $F_{\text{flex}}(S, P1) = \frac{6}{6} = 100\%$ and $F_{\text{flex}}(S, P2) = \frac{2}{6} = 33\%$.

11.4.2.1.2. *A posteriori* performance measures

To measure the capacity of a reactive algorithm P to provide good performance, we propose two *a posteriori* performance measures, F_{perf}^1 and F_{perf}^2 defined as follows

$$F_{\text{perf}}^1(S, P) = \frac{C_{\max}(S, P) - bC_{\max}(S)}{bC_{\max}(S)} \quad (11.9)$$

$$F_{\text{perf}}^2(S, P) = \frac{\bar{T}_w(S, P) - b\bar{T}_w(S)}{b\bar{T}_w(S) + 1} \quad (11.10)$$

where $bC_{\max}(S)$ and $b\bar{T}_w(S)$ are respectively the best makespan and weighted sum of tardiness of solution S .

11.4.2.2. Proposed algorithms

The algorithms that we propose all work the same way. They use priority rules in a lexicographical way. At a given moment t , algorithms schedule job i^* maximizing $\text{PRIOR1}(i, t)$ among the available jobs i . When two or more jobs are in competition, the job maximizing $\text{PRIOR2}(i, t)$ is selected. What differentiates these algorithms are priority functions $\text{PRIOR1}(i, t)$ and $\text{PRIOR2}(i, t)$ and the definition of job availability. A job is available if all its predecessors have already been scheduled and if it fulfills the selected scheduling constraints. Even though the solution was calculated *a priori* by considering semi-active schedules, we can decide for performance reasons to only build active or non-delay schedules in real time. Note that active and non-delay schedules are semi-active schedules.

We propose different priority rules intended for optimizing one of the *a posteriori* quality measures:

– $F_{\text{flex}}(S, P)$ is increasing according to the number of alternatives proposed in each phase. To maximize this number, we propose a priority rule which selects the available job that will free up the largest number of successors. The algorithms connected to this priority rule are called *Flex1_SA*, *Flex1_A* and *Flex1_ND* where *SA*, *A* and *ND* indicate the type of scheduling considered: semi-active, active and non-delay. If two jobs are in competition, the job with the shortest, earliest start date (refreshed) is selected.

– It is also advisable to retain some flexibility over sequences throughout execution. In order to do this, we propose a priority rule which selects the available job maximizing the number of remaining non directed arcs. The algorithms connected to this priority rule are called, as in the previous point, *Flex2_SA*, *Flex2_A* and *Flex2_ND*. If two jobs are in competition, the job with the shortest earliest start date (refreshed) is selected.

– In order to favor the main performance criterion, which for us is the weighted sum of tardiness, we use known priority rules for this criterion [MOR 93]. In this case, we have

$$\text{PRIOR1}(i, t) = \frac{w_j}{p_j} \exp \left(- \frac{\max \{d_j - (t + p_j), 0\}}{kp_{av}} \right) \quad (11.11)$$

where p_{av} is the average job execution times and k , whose value is generally chosen around 3, is a parameter expressing how the priority of a job increases when its margin decreases.

The algorithms proposed are noted as *Perf_SA*, *Perf_A* and *Perf_ND*.

11.4.3. Reactive algorithm with disruptions

During the occurrence of a disruption, an analysis module is used. Two cases are considered:

- *Case of machine breakdown*: the module uses results developed in [ALO 02a] to calculate upper bounds on the increase for the worst weighted sum of tardiness, and to propose, if possible, one or more decisions to minimize this increase without violating the initial partial sequence. The increase of the worst makespan can be calculated in polynomial time [ALO 04]. To minimize this increase, the module can propose a shift to non-delay control.

- *Case of late component arrival*: we presume that the control algorithm chooses to schedule a job T_j with one of its components being late. This job can only be executed from a date $r_j^m > t$. If an available job T_i exists such that the arrival of its components is not disrupted, then scheduling T_i at moment $\max\{r_i, t\}$ does not increase the worst weighted sum of tardiness of the current solution. If such a job does not exist, the disruptions can then be considered as failure and an upper bound of the increase can thus be calculated.

Generally, if the upper bound of the expected increase over the weighted sum of tardiness is estimated as small, the module proposes the scheduling of the job minimizing this increase. If that is not the case, the module can propose either to enrich the partial sequence by adding arcs in order to limit performance deterioration in the worst of cases, or to execute global rescheduling, i.e. recalculate a new solution for remaining jobs.

11.5. Experiments and validation

The validation of our approach required three phases of experiments. In the first phase, we experimented with the capacity of the proactive algorithm to explore the different research spaces, defined by the qualitative measure of flexibility over sequences, and to provide solutions offering an acceptable compromise between *a priori* measures of performance and flexibility. In the second phase, we considered that the workshop is not disrupted and we measured the aptitude of control algorithms to use the flexibility introduced by the proactive algorithm while providing good performances at the end of execution. In the third phase, we compared our approach to a reactive predictive approach in the context described in the introduction to this chapter.

In what follows, we summarize the results obtained in [ALO 02b, ALO 02a, ALO 03]:

- For all solutions S calculated by the proactive algorithm, we have $0.8\bar{T}_w^* \leq b\bar{T}_w(S) \leq 1.04\bar{T}_w^*$, where \bar{T}_w^* is the weighted sum of tardiness given by the best scheduling calculated with heuristics known for this criterion such as ATC and X-RM. This means that each solution S contains at least one schedule with very good performances and that we can even increase the performance by approximately 20% in relation to the best solution found off-line (obtained by a heuristic).
- The proactive algorithm offers a good compromise between performance and flexibility in the case where job start dates are grouped in time. When these start dates are spread out, job permutation must be limited in order to obtain good performances.
- In the absence of disruptions, procedures *Flex1_ND* and *Perf_ND* dominate, in most cases, the other procedures and provide an acceptable compromise between *a posteriori* measures of quality.
- In the presence of disruptions, our approach is superior to the reactive predictive approach for disruptions on availability of raw material (main and intermediate components) with low or average amplitudes and for a reasonable number of breakdowns. In this comparison, we have given the predictive solution a flexibility over time globally equivalent to the flexibility included in the proactive solution in order to obtain very similar intermediate product storage costs for both approaches.

11.6. Extensions and conclusions

In this chapter, we considered the scheduling problem for activities in a workshop made up of only one workstation for the assembly of components coming from downstream workshops and components purchased from suppliers. In order to minimize the effect of disruptions which can occur from the different players in the logistics chain, we proposed a proactive-reactive approach to anticipate these disruptions. At the proactive level, a genetic algorithm calculates one (or more) solution characterized by a partial order between activities and a type of schedule. Such a solution, said to be flexible, then represents several schedules with common properties in order to easily move from one schedule to another. This constitutes flexibility over sequences and over time that the reactive algorithm is supposed to use knowingly to control the workshop. Control decisions involve the execution of jobs and reaction to possible disruptions. We demonstrated using experiments the superiority of our approach compared to traditional predictive-reactive approaches.

The results obtained for single machine scheduling problems gave us some insight to extend our approach to the flow-shop scheduling problem [ALO 09].

In our approach, disruptive characteristics, such as frequency and time, are not known beforehand. It would be interesting to find out how to integrate a possible knowledge of these characteristics as a research guide for flexible solutions.

11.7. Bibliography

- [ALO 01] ALOULOU M.A. and PORTMANN M.C., “Incorporating flexibility in job sequencing for the single machine total weighted tardiness problem with release dates”, *Proceedings of the 10th Annual Industrial Engineering Research Conference*, May 2001, CD-ROM.
- [ALO 02a] ALOULOU M.A., “On the reactive scheduling design using flexible predictive schedules”, *Proceedings of IEEE SMC’2002*, Hammamet, October 2002, CD-ROM.
- [ALO 02b] ALOULOU M.A., Structure flexible d’ordonnancements à performances contrôlées pour le pilotage d’atelier en présence de perturbations, PhD Thesis, Institut National Polytechnique de Lorraine, December 2002.
- [ALO 03] ALOULOU M.A. and PORTMANN M.C., “An efficient proactive reactive approach to hedge against shop flow disruptions”, *Proceedings of MISTA Conference 2003*, August 2003.
- [ALO 04] ALOULOU M.A., KOVALYOV M. and PORTMANN M., “Maximization in single machine scheduling”, *Annals of Operations Research*, vol. 129, p. 21–32, 2004.
- [ALO 07] ALOULOU M., KOVALYOV M. and PORTMANN M.-C., “Evaluating flexible solutions in single machine scheduling via objective function maximization: the study of a computational complexity”, *RAIRO Operations Research*, vol. 41, p. 1–18, 2007.
- [ALO 09] ALOULOU M. and ARTIGUES C., “Flexible solutions in disjunctive scheduling: general formulation and study of the fow-shop case”, *Computers and Operations Research*, 2009, forthcoming.
- [BRI 91] BRIGHTWELL G. and WINKLER P., “Counting linear extensions”, *Order*, vol. 8, p. 225–242, 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”, *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, p. 2922–2927, October 14-17 1996.
- [GOL 89] GOLDBERG D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.
- [GOT 02] GROUPE FLEXIBILITÉ DU GOTHA, “Flexibilité et robustesse en ordonnancement”, *Bulletin de la ROADEF*, no. 8, p. 10–12, 2002, full version available on <http://wwwmath.univ-bpclermont.fr/sanlavil/FRO.html>.

- [LAW 73] LAWLER E.L., "Optimal sequencing of a single machine subject to precedence constraints", *Management Science*, vol. 19, p. 544–546, 1973.
- [LAW 93] LAWLER E.L., LENSTRA J., RINNOOY KAN A. H.G. and SHMOYS D.B., "Sequencing and scheduling: algorithms and complexity", in GRAVES S., RINNOOY KAN A. and ZIPKIN P. (Eds.), *Handbooks in Operations Research and Management Science, Volume 4: Logistics of Production and Inventory*, North-Holland, Amsterdam, 1993.
- [MOR 93] MORTON T.E. and PENTICO D.W., *Heuristic Scheduling with Applications to Production Systems and Project Management*, Wiley, New York, 1993.
- [POR 98] PORTMANN M.C., VIGNIER A., DARDILHAC C.D. and DEZALAY D., "Branch and bound crossed with GA to solve hybrid flowshops", *European Journal of Operational Research*, vol. 107, p. 389–400, 1998.

Chapter 12

Stabilization for Parallel Applications

The recent development of new parallel and distributed platforms based on the interconnection of a large number of standard components has significantly changed the parallel processing field. The most important point for a more effective use of such systems is the management and optimization of resources, particularly scheduling. This consists of allocating the tasks of a parallel program to processors on the platform and determining at what time the tasks will start their execution.

Now more than ever, handled data are subject to uncertainties and/or disturbances, and thus it is almost impossible to have a precise prediction of the input parameters of the scheduling problem, particularly those related to communications. We propose in this chapter a brief survey of the existing approaches dealing with disturbances in the context of the new parallel and distributed systems. Then, we present different methods to study scheduling algorithms with the ability to absorb any perturbation on the data and if necessary to develop new mechanisms to adapt these algorithms to the new parameters during the execution. More specifically, to deal with disturbances, we focus on a partially on-line approach (proactive/reactive approach) which start from an initial solution computed with estimated data and correct it on-line depending on the values of actual data, that is, the stabilization approach.

12.1. Introduction

In this chapter we deal with the problem of the analysis and design of efficient scheduling algorithms in the domain of parallel and distributed systems. This domain has changed considerably in the last decade with systems processing many new features that should be taken into account when optimizing the performance. The most important point for a more effective use of such systems is the management and optimization of resources, particularly scheduling. This consists of allocating the tasks of a parallel application to processors on the platform and determining at what time the tasks will start their execution. In most cases, this application is described in a high-level programming language from which we extract (more or less automatically) the tasks, with their relations of interdependence. The scheduling is determined before execution by more or less sophisticated methods, based on estimated data of the tasks and their structures. However, the handled data can be strongly modified at run-time by many unpredictable phenomena. The effects of disturbances on the data may impact the system's efficiency, eventually leading either to an unfeasible situation or to the generation of opportunities that improve its performance. Therefore, the problem is how to find a good schedule in such a context. Thus, it is essential to develop new mechanisms for controlling (and if necessary adjusting) the algorithm to guarantee reasonable performances.

This chapter is organized as follows. In section 12.2 we give a brief introduction to the domain of parallel and distributed systems and state the computing model. Next, in section 12.3, we review the main existing approaches dealing with uncertainties in the actual parallel systems. Then, in section 12.4, we focus on partially on-line approaches and we describe the stabilization process. Section 12.5 describes two results for stabilization, a first one called the *PRCP** algorithm and a second algorithm that guarantees a *strong* stabilization. In section 12.6 we present a new, intrinsically stable scheduling algorithm, that is, one that is able to absorb the bad effects of disturbances occurring at runtime. Finally, before concluding this chapter and giving some perspectives, we present some experimental results that assess the good behavior of the new intrinsically stable algorithm.

12.2. Parallel systems and scheduling

In this section, we describe in detail the domain of parallel processing which is the target for promoting the stabilization approach.

12.2.1. Actual parallel systems

Since the end of the 1990s, the domain of parallel processing has changed considerably. The enormous advances in computing capability, the integration of

computing and communications, and the reduction in the cost of these technologies led to an increase in number and size of parallel and distributed systems. Traditional parallel architectures have been replaced by large collections of standard computing components that communicate with each other over local or large area networks [CUL 99]. A huge number of new computing platforms have been installed all over the world because they are a low cost alternative to super-computers. The concept of grid computing can be used in different contexts, depending on the type of components or connections. The first type of grids correspond to a medium size and are composed of rather homogenous processors, typically several hundred CPUs locally connected by a fast network [BUY 99]. They are usually under the administration of a single operating system. Computational grids may be composed by more heterogeneous and distant processor units (some can even be super-computers) [FOS 99]. Several operating systems run simultaneously on different machines. Such systems are stable in the sense that any computing unit participating to the grid has to be identified before. Global computing goes one step further. It corresponds to a very large number of relatively small computational units (several thousands) that can appear or disappear at any moment [GER 01]. Nowadays, the computational capabilities of stand-alone computers have increased thanks to the dual-core or multicore architectures. With the introduction of this type of architectures even mainstream PCs have become parallel systems.

All these new computing platforms are characterized by many new features, for example, large communication delays with relatively slow connections in regard to the computations, hierarchy in computing and communication media, heterogeneity and volatility of the resources. They all have in common a larger complexity than traditional parallel machines. With various degrees, the consequences are an increased difficulty in predicting the parameters that are the input of the scheduling problem, particularly for communications. In this chapter, we consider homogenous processors for a simplified presentation, although the results may be extended to heterogeneous platforms.

12.2.2. Definitions and notations

In this section, we present the scheduling problem in detail. This problem deals with the optimal assignment of a set of tasks to processing elements in distributed systems in such a way that the completion time of all tasks is minimized. In the context of new parallel and distributed systems several criteria can be optimized. In this chapter, we focus on the minimization of the *schedule length* or *makespan* (i.e., maximum completion time over all the processors) [LEU 04].

We consider the computational model formalized by Rayward-Smith [RAY 87] and Papadimitriou and Yannakakis [PAP 90], known as the *delay* model. It is a classical model where the communication between tasks allocated on different processors is explicit and can be overlapped by local computations. Communications between tasks executed on the same processor are neglected (this is known as locality assumption). A parallel application is usually modeled by precedence task graphs. It is represented by a *directed acyclic graph* $G = (V, E)$, where $V = \{1, \dots, n\}$ is the set of tasks to be scheduled, and $E \subseteq V \times V$ is the set of edges. Each edge $(i, j) \in E$ represents a precedence constraint that means “the results of task i must be available before j starts its execution”, Figure 12.5 give an illustration of such a graph. We denote by $\text{PREC}(i)$ and $\text{SUCC}(i)$ the set of predecessors and successors of task i , respectively $\text{PREC}(i) = \{j \in V; (j, i) \in E\}$ and $\text{SUCC}(i) = \{j \in V; (i, j) \in E\}$.

The processing time of a task $j \in V$ is denoted by $p_j \in \mathbb{N}^+$. It takes a time $c_{ij} \in \mathbb{N}^+$ to transfer data from task i to task j , $\forall (i, j) \in E$, which is zero if both tasks are scheduled into the same processor. However, some disturbances can affect the communications at run-time. In this case, for a given graph G , we denote by \tilde{G} the graph whose structure is the same as that of G , but differs in the cost c_{ij} associated with the edges. This cost is denoted \tilde{c}_{ij} (more generally, we will denote \tilde{x} the disturbed value of parameter x).

Let us consider an application composed of n tasks to be processed on m identical processors. If $m \geq n$, then the number of available processors is unrestricted. A schedule is defined as a pair of applications, namely, $\sigma : V \rightarrow \mathbb{N}^+$ and $\pi : V \rightarrow \{1, \dots, m\}$, where $\sigma(j)$ represents the starting time of task j and $\pi(j)$ provides the processor on which j is executed. The feasibility of the schedule means the respect of the precedence constraints which are guaranteed by the following relations:

- $\forall (i, j) \in E$, $\sigma(j) \geq \sigma(i) + p_i$ if i and j are allocated on the same processor ($\pi(i) = \pi(j)$) and, $\sigma(j) \geq \sigma(i) + p_i + c_{ij}$ otherwise ($\pi(i) \neq \pi(j)$). Task j cannot be executed before task i has been completed,

- for any pair of tasks $\{i, j\}$, if $\pi(i) = \pi(j)$, then $\sigma(i) + p_i \leq \sigma(j)$ or $\sigma(j) + p_j \leq \sigma(i)$. Each task is allocated to only one processor and one processor cannot execute more than one task at a time,

- preemption is not allowed: the execution of a task cannot be interrupted and resumed at a later time.¹

1. Preemption can be an interesting issue for scheduling independent jobs on uniformly related parallel machines on-line [SHM 95].

Let (σ, π) be a feasible schedule for $G = (V, E)$. The *completion time* of each task $j \in V$ is defined by $C_j \equiv \sigma(j) + p_j$. The *length* or *makespan* of (σ, π) is the maximum completion time (denoted by C_{\max}) of the task. Formally, $C_{\max} \equiv \max_{j \in V} C_j$. The objective is to minimize C_{\max} .

The scheduling problem is known to be NP-hard in its simplest version (without communications) – for more details we refer the reader to [ULL 75]. Many approximation methods have been developed for variants of this problem and some of them have been implemented in operational parallel environments of old generation parallel systems [YAN 92].

The scheduling problem has usually been seen as a function of known and reliable information. Most approaches that have been developed are mainly deterministic, that is, they are based on nominal or estimated values for all the parameters, thus implicitly assuming that a *baseline schedule* will be executed exactly as planned. However, this assumption is rather idealistic since many unpredictable events continually occur. Scheduling problems involve data, coming from different sources, and which vary rapidly over time as resource availabilities. Data may be ambiguous, outdated or inaccurately predicted before the problem is solved. Nevertheless, in the context of new computing platforms, the processing time of tasks can be approximately estimated depending on the processor where the tasks will be executed. It is claimed, however, that communication delays are subject to many unpredictable events that are hard to anticipate because of a highly disturbed context of the actual execution of parallel applications. In this case, building an assured theoretical model for such complex systems is a very intricate problem and the analysis is in general intractable. Therefore, the problem is finding a good schedule that takes into account only a partial knowledge.

12.2.3. Motivating example

In this section, we give an example of the impact of disturbances when scheduling an application with perturbations in the communication delays.

As a consequence of the inherent uncertainty when deciding a baseline schedule in new parallel systems, disruptions may appear at execution time affecting the implementation of the schedule. Disruptions may be complex, multiple in nature, and appear at any moment over the span of the schedule. Several sources of disturbances in communication delays exist when scheduling an application. For example, in many models, communication delays only depend on the source and destination

tasks, not on the communication network. The general assumption is that such a network is logically fully connected, and that the lengths of the links are equal. This is rarely the case for a real machine: the network topology and the contention of the communication links may largely influence the delays, not to speak of communication failures. Another example is where the communication takes place over a network where there may be contention for transmission, so communication delays are non-deterministic.

We focus in this chapter on the impact of disturbances on the communication delays which is the most significant performance factor in parallel and distributed systems. We illustrate below the effects of disturbances on a simple example.

Let us consider the pre-allocated precedence task graph depicted in Figure 12.1. It is composed of 9 tasks of unit execution time and all the communications have an estimated value of 1.

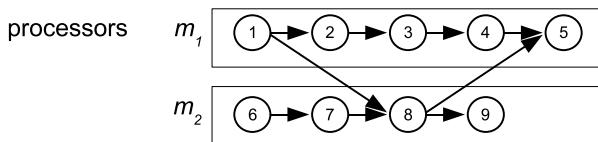


Figure 12.1. A precedence task graph with its allocation

Let us assume that this pre-allocated precedence task graph is scheduled on two processors m_1 and m_2 . Now, let consider the optimal schedule depicted as a Gantt chart in Figure 12.2(a). There are two effective communications² between the tasks 1 and 8 (c_{18}), and 8 and 5 (c_{85}). The schedule length is $C_{\max} = 5$.

Let us now imagine that the actual durations of the effective communications (i.e., communications after disturbances) are $\tilde{c}_{18} = 1 + \epsilon_1$ and $\tilde{c}_{85} = 1 + \epsilon_2$.

A slight increase in the communication cost c_{18} results in delaying the starting time of task 8 by ϵ_1 and thus the communication between tasks 8 and 5 is also delayed by the same amount of time. Another disturbance ϵ_2 occurs in the communication c_{85} . Finally, it leads to an actual makespan $\tilde{C}_{\max} = 5 + \epsilon_1 + \epsilon_2$, as depicted in Figure 12.2(b). Of course, if the disturbances are greater, a change of allocations would be profitable.

2. A communication is called effective between two communicating tasks if they are scheduled on different processors.

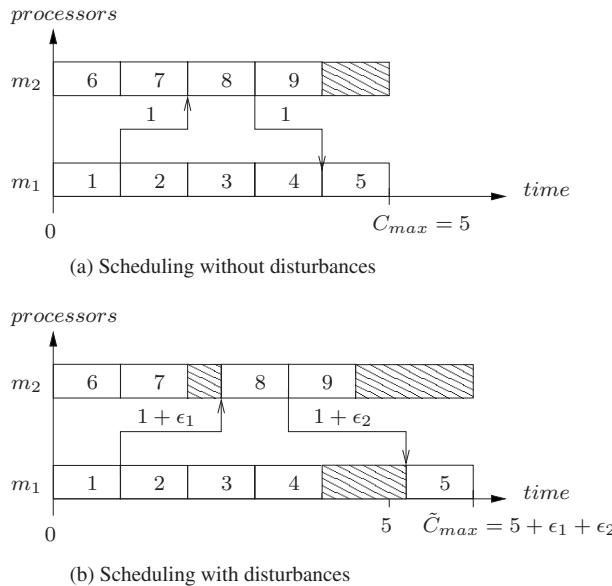


Figure 12.2. Impact of disturbances on the communications

Let us note from this example that the cumulative effect of the disturbances can be highly significant if several disturbances occur, which is not rare in practice. This is because the cumulative effect relies on communications occurring in opposed directions between both processors. This can of course be generalized to any number of processors.

12.3. Overview of different existing approaches

In order to contextualize the stabilization approach, we briefly recall the different possible approaches dealing with uncertainties in the scheduling problem. Most of them have been presented and discussed in other chapters of this book.

Fundamentally, the main difference between the possible approaches is to consider if a solution can or cannot be modified during the execution, when the actual values are known. If a solution cannot be modified, we deal with static or *a priori* approaches, and the problem here is to construct a “good” solution that will not be impacted too much by the disturbances. If a solution can be modified during the execution we are dealing with dynamic approaches. More specifically:

- Static or robust approaches called *a priori*: a schedule is fully computed before its execution with the estimated data.

– Dynamic or *on-line* approaches: in such an approach the schedule is computed at execution time using simple priority rules when the actual data are known. It is difficult to take advantage of partial knowledge of the problem.

In addition to these two approaches, *sensitivity analysis* studies the performance change of a given solution, due to a change in the data, that is, this approach analyzes the variation of the solution relative to the disturbances on the estimated values which occur at run-time. It is commonly used to assess the robustness of a schedule to perturbations in the model's specifications or input data [PEN 01].

Between the *a priori* and on-line approaches it is natural to consider an intermediate approach, the *partially on-line* approach. The principle is to compute an initial schedule with the estimated data and, depending on the on-line disturbances, to adjust the schedule at run-time. From this partial knowledge, a scheduling solution can be computed statically and then, potentially corrected at run-time. This is the main idea behind the stabilization approach which will be described in the following sections.

12.4. The stabilization approach

Compared to predictive approaches, stabilization is a partially on-line approach. As we mentioned earlier, partially on-line (flexible) is an intermediate approach between *a priori* and pure on-line approaches. In this approach, it is possible to adapt a solution on-line once the actual data are known, in order to achieve a better performance than the initial solution computed with the estimated data, that is, the goal is to reduce the disturbance impact on the performance of the solution.

Schedule stabilization has been addressed in two main domains: parallel processing and real-time systems. In this work, we focus on the stabilization approach in the field of parallel processing. For more details about stabilization in real-time systems we refer the reader to [MAN 67].

12.4.1. Stabilization in processing computing

For the user, one of the goals of parallel processing is to reduce the computational time of any application and thus to minimize the makespan. We define a stable schedule if the final makespan (obtained after running the algorithm on disturbed data) is close to the initial makespan computed before the execution, without referring to the optimal value of the actual instance; that is, the aim is to compute

a makespan as close as possible to the initial schedule. We illustrate this definition using Figures 12.3 to 12.4. Consider a set of instances \mathcal{P} (Figure 12.3), an instance $I \in \mathcal{P}$ (estimated instance) and a schedule σ . C_{\max} is the makespan computed by schedule σ over the instance I and C_{\max}^* is the optimal makespan.

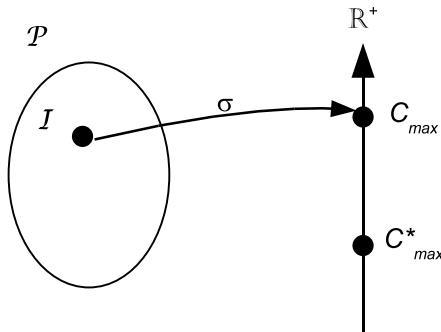


Figure 12.3. Scheduling instance I without disturbances

If any disturbance occurs at run-time that transforms the instance I into a new instance \tilde{I} , Figure 12.4(a), σ is stable if the makespan \tilde{C}_{\max} (the makespan of the actual instance \tilde{I}) remains close to the initial makespan C_{\max} estimated without disturbance. Note that the optimal solutions C_{\max}^* and \tilde{C}_{\max}^* cannot be so close, Figure 12.4(b). Let us note that this definition of stability is coherent with the mathematical definition of stability.

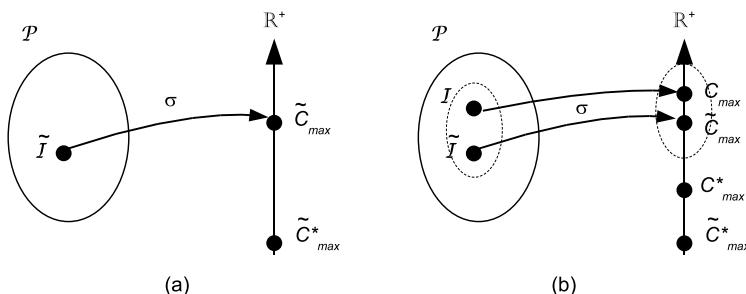


Figure 12.4. a) Applying σ on a disturbed instance; b) scheme of a stable solution

Based on this definition, several works have been proposed to develop algorithms that compute stable schedules. In general, the stabilization (also called stabilization process) is a 2-phase approach based on on-line sequencing after a statically fixed

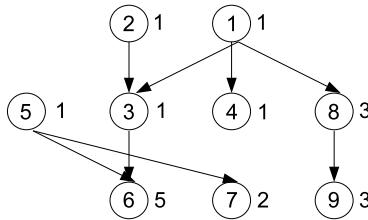
task allocation; that is, in the static phase, some mechanisms are used to stabilize the application (i.e. to include safety in the baseline schedule in order to absorb the anticipated disturbances as effectively as possible) and to define a procedure to react to disturbances in the dynamic phase that cannot be absorbed by the baseline schedule. We describe these two phases below.

Let us consider a predicted schedule obtained by any algorithm based on the initial precedence task graph (that is, with estimated data). The principle of the stabilization consists of adjusting this schedule on-line if any disturbances occurring at run-time affect communication delays. This hypothesis assumes that the scheduler is able to adapt or adjust the schedule on-line. With the scheduler, from the theoretical point of view, all the permutations are possible, that is, if any perturbation affecting one communication makes one task unavailable at the estimated date, then the scheduler can schedule another task instead of the predicted one being executed at this date. If necessary, the scheduler changes the processor allocation between tasks. It might be interesting to change the allocation because we can adjust the baseline schedule as we want. Unfortunately, the permutation between tasks could lead to an important degradation instead of makespan minimization; that is, an on-line adaptation cannot ensure the stability of the estimated makespan. For instance, if a high priority task is not executed at the time determined by the static schedule, due to a slight perturbation, the scheduler will execute another ready task, which is equivalent to the fact that the priority list of execution is changed. Consequently, the makespan can increase beyond what could have been achieved with a static execution. This is the case for the list scheduling algorithms and is known as the *Graham anomalies* [GRA 66]. In practice, we assume that the scheduler permutes only tasks allocated on the same processor (however, this assumption does not prevent all anomalies in the schedule). We shall briefly illustrate an example of the bad effects of some on-line permutations of tasks.

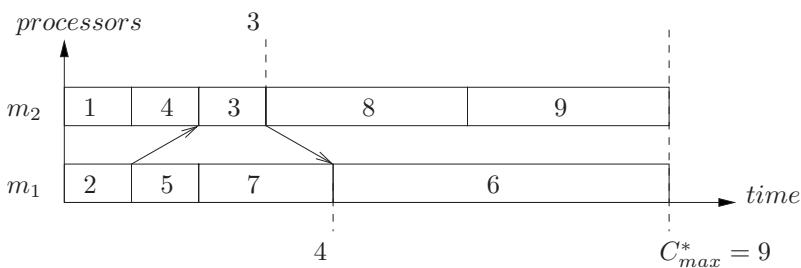
12.4.2. Example

First, consider the precedence task graph depicted in Figure 12.5: the nodes represent tasks and edges represent precedence constraints, as described earlier in this chapter. The corresponding processing time is given next to each node and all the communications have an estimated value of 1. In the original formulation, Graham focused on computations and did not consider communications. We present below an adaptation of these anomalies for communications.

Figure 12.6 shows a list schedule obtained for the graph of Figure 12.5 on two processors (the schedule is optimal). Consider now an on-line execution of this

**Figure 12.5.** Precedence task graph

schedule: the list of priorities of the task executions is the sequential order on each processor in the static execution, that is, $L_2 = \{2, 5, 7, 6\}$ on processor m_1 and $L_1 = \{1, 4, 3, 8, 9\}$ on processor m_2 .

**Figure 12.6.** Initial schedule

Imagine now that the duration of task 4 decreases by ϵ (it is equivalent that the communication between tasks 2 and 3 increases by ϵ). When the execution of task 4 is completed, task 3 is not ready to schedule because of the communication originating from task 2. Thus, the scheduler schedules the first ready task³ of the list, in this case task 8 and the execution of task 3 is delayed until the completion of task 8. The on-line schedule computed is given in Figure 12.7 with a makespan increased to $\tilde{C}_{\max} = 12 - \epsilon$. On the contrary, the makespan of the static execution is $C_{\max}^* = 9$. As task 3 needs to communicate with task 6, so any delay on the execution of task 3 will delay the completion time of task 6. For this example, task 3 is a critical (high-priority) task. Thus, in this case, on-line adaptation of the schedule deteriorates the performance of the schedule.

3. A task i is ready on processor $\pi(i)$ at time t if all data from its predecessors have arrived in $\pi(i)$ and can be immediately executed on that processor at that time.

It is clear that in some cases, it is preferable to wait for a critical task to become ready like in the static schedule instead of executing another task. Then, the problem is to prohibit the bad permutations of tasks. The addition of pseudo-constraints might alleviate this problem, that is, the schedule is stabilized by adding precedences or *pseudo-edges* between some tasks allocated inside the same processors. In the following sections, we will describe the stabilization process in more detail.

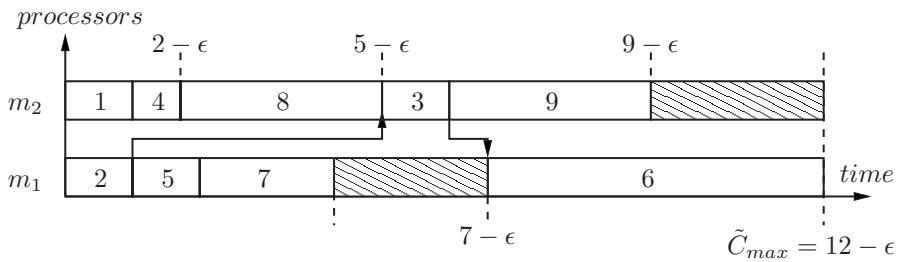


Figure 12.7. On-line execution of the initial schedule

12.4.3. Stabilization process

The principle underpinning the stabilization process consists of allowing only the on-line permutations which can improve the makespan, that is, to prevent the scheduler from bad permutations between tasks occurring at run-time and leading to an increase in the makespan. As the final objective is to minimize the computational time, adding some mechanism to select the right permutations (i.e., the permutations allowing us to decrease the makespan) can be costly and can result in a higher computational time. So, the idea is to determine the right permutations before the execution.

12.5. Two directions for stabilization

The main approaches for stabilization are based on a static process that is achieved before beginning the execution. The goal is to determine (and to forbid) the permutations leading to anomalies on the initial graph. As we have already mentioned, the stabilization process is considered a 2-phase approach, as follows:

- compute a baseline schedule and stabilize it,
- execute the schedule on-line.

Let us emphasize again that the schedule is stabilized by adding pseudo-edges between some tasks allocated into the same processors. This is a static phase. In the following section, we survey two existing algorithms used for stabilizing an application.

12.5.1. The $PRCP^*$ algorithm

Moukrim *et al.* proposed in [MOU 99] a partially on-line stabilization algorithm (called $PRCP^*$) in the case of join and fork graphs. The algorithm is based on a critical path to cope with the possible disturbances. The disturbances concern communication delays, and in this model an estimation of the communication delays is known at compilation time. However, due to network contention, link failure, etc., communication delays are disturbed at run-time. The authors claim that because of the lack of accurate estimation of the communication delays, building a full-fledged schedule at compile time is inappropriate. On the other hand, building the schedule completely at run-time is also unsatisfactory. Therefore, a trade-off between these two approaches is proposed. This approach (proactive) is decomposed into two main phases, as follows:

- **1.1** compute an off-line schedule based on the estimated communication delays and **1.2** compute a partial order by adding precedences between tasks assigned to a same processor. The goal of this step is to stabilize the schedule obtained in the previous step.
- **2** execute on-line the static schedule (i.e., the assignment of step 1.1) considering the pseudo-edges (that is, the partial order) added in step 1.2.

The algorithm works as follows. The schedule of step 1.1 is built by any static algorithm. The authors used the well-known ETF⁴ (*earliest task first*) algorithm [HWA 89] based on critical path priority (denoted ETF/CP). The priority list is defined as follows: for each task i compute $L^*(i)$, the longest path to a final task, including processing times and communication delays (between tasks to be processed on different processors), the processing time of the final task but not the processing time of i . Thus, the priority of i is proportional to $L^*(i)$. The heuristic that sequences ready tasks using the above priority is called RCP^* in [YAN 93].

Suppose that two tasks i and j are assigned to the same processor, with the partial order of step 1.2 being obtained from the two following conditions:

- 1) task i has higher priority than j ,
- 2) task i is sequenced before j in the schedule off-line (i.e., the schedule obtained in step 1.1).

4. ETF is a list scheduling algorithm and considers a bounded number of processors.

If the two previous conditions hold, then a pseudo-edge is added from i to j . This will avoid a disturbance that leads to executing j before i at run-time.

Finally, in phase 2, RCP^* is again used as sequencing on-line policy to obtain the complete schedule. The authors called the resulting algorithm $PRCP^*$ for partially on-line sequencing with RCP^* .

We illustrate the execution of the algorithm by the following example. Let us consider first the pre-allocated precedence task graph depicted in Figure 12.8(a). It is composed of 9 tasks. The corresponding processing time is given next to each node and all the communications have an estimated value of 1. Figure 12.8(b) shows one possible schedule obtained at compile time by ETF/CP in step 1.1. Its completion time is $C_{\max} = 6$. Figure 12.9 depicts the pre-allocated precedence task graph after the stabilization, i.e., after adding the pseudo-edges (dashed edges). The following pseudo-edges are added by PRCP*: for processor m_1 , between tasks 2 and 6, and between tasks 2 and 7, while for processor m_2 , between tasks 3 and 8.

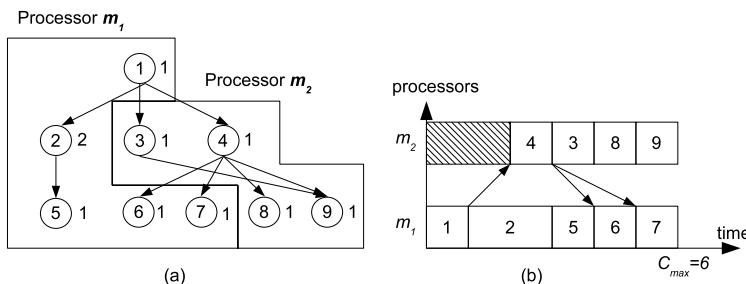


Figure 12.8. a) Pre-allocated precedence task graph;
b) schedule computed in step 1.1 by ETF/CP

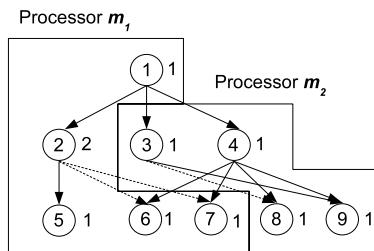


Figure 12.9. The pre-allocated precedence task graph after stabilization: step 1.2.
Dashed edges represent the pseudo-edges added by $PRCP^*$

Let us now suppose that the actual duration of the communication between the tasks 1 and 4 is $\tilde{c}_{14} = c_{14} \cdot (1 + \varepsilon)$. The makespan of a static execution of the schedule computed by ETF/CP is equal to $6 + \varepsilon$. However, $PRCP^*$ produces a makespan equal to 7 because of the small disturbance of c_{14} leads to execute task 3 before task 4 at execution time (step 2), that is, at time 2 task 4 is not ready for processing, but task 3 is. No additional precedence was added between tasks 4 and 3 as they have the same priority. Thus, $PRCP^*$ executes task 3 before task 4 and increases the actual completion time, as in Figure 12.10. Let us note that for this example the schedule obtained at on-line execution without the stabilization process is also equal to 7.

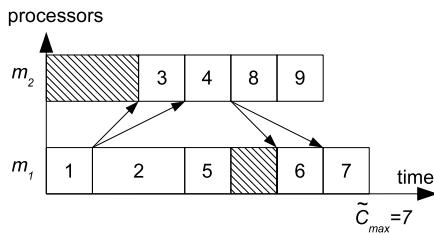


Figure 12.10. Schedule obtained by $PRCP^*$ after the stabilization: step 2

Clearly, $PRCP^*$ fails here because it only takes $L^*(i)$ into account, but not the number of successors of i . However, examples can be easily built for which taking this number into account (or using any other priority computation) does not guarantee stability. Nevertheless, Moukrim *et al.* showed that the $PRCP^*$ algorithm is optimal for a fixed assignment when the task graph is a fork or join graph.

In order to avoid any degradation of the baseline scheduling, Gupta *et al.* in [GUP 03] investigated means of identifying a minimum set of additional edges in the context of partially on-line scheduling. The aim is to protect the baseline scheduling from performing badly due to disturbances. Moreover, this construction guarantees a result at least as good as the result obtained for the initial static scheduling, that is, the *strong stabilization*. We present this work in the following section.

12.5.2. Strong stabilization

Gupta *et al.* [GUP 03] proposed another stabilization algorithm based on a similar stabilization principle [MOU 99]. The authors consider the problem of scheduling, precedence task graph with disturbances in processing time and communication delays. Their solution starts from a static schedule computed with the estimated data

by any well-known static algorithm, like ETF or DSC⁵ (*dominant sequence clustering* [YAN 94]). Based on this schedule, the authors add some pseudo-constraints to the precedence graph.

DEFINITION 12.1.— *A communicating task is a task of where at least one of the immediate successors is executed on a different processor.*

The aim of the algorithm is to reduce the disturbance impact, but at the same time to guarantee that the makespan obtained at on-line execution after the stabilization is smaller than or equal to the makespan obtained with static execution on the actual instance. They called this condition *strong stabilization*.

Let σ be the schedule computed at static execution, and let \tilde{C}_{\max} be the actual makespan, that is, the disturbed makespan obtained by executing σ on the actual instance. Let \tilde{C}'_{\max} be the makespan obtained after the stabilization. The condition of the strong stabilization holds if

$$\tilde{C}'_{\max} \leq \tilde{C}_{\max}$$

whatever the instance processed. For a given schedule σ , Gupta *et al.* introduced the concept of permutable tasks, which is defined as follows: task i is permutable *if and only if* the following two conditions holds:

- task i communicates with a task j scheduled on another processor,
- there exists at least one task j that is scheduled in σ after task i on the same processor, and i and j are independent.

PROPERTY 12.1.— *The only pseudo-edges that are necessary to add are those going from a permutable task i to the set of tasks with which i could permute.*

The principle behind strong stabilization is to prevent permutations between tasks, which can increase the makespan. Gupta *et al.* proved that the minimal set of pseudo-edges to guarantee the strong stabilization condition is the pseudo-edges emanating from the permutable tasks.

Finally the algorithm is expressed as follows:

- *Phase 1.* Compute an off-line schedule using ETF or DSC. For each permutable task i , add pseudo-edges between i and the set of tasks with which i could permute.

5. DSC is a clustering-based scheduling algorithm and considers unbounded number of processors.

– Phase 2. Each processor executes the tasks according to a ready-list maintained locally.

Let us illustrate the execution of the algorithm with the following example. Consider again the preallocated precedence task graph depicted in Figure 12.8(a) and the associated schedule. Only task 4 satisfies the conditions of a permutable task, that is, task 4 is a permutable task and the task this could permute is task 3, so the algorithm adds only the pseudo-edge between task 4 and 3; see Figure 12.11.

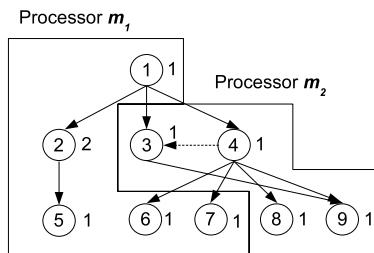


Figure 12.11. The pre-allocated precedence task graph after stabilization. The dashed edge represents the pseudo-edge added by the strong stability algorithm

Suppose now a disturbance occurs between task 1 and 4; the actual communication is $\tilde{c}_{14} = c_{14}(1 + \varepsilon)$. Even with this disturbance at on-line execution, task 4 is scheduled before task 3. Thus, the completion time of the schedule after the stabilization is $\tilde{C}'_{\max} = 6 + \varepsilon$, which is equal to the makespan at static execution after disturbances $\tilde{C}_{\max} = 6 + \varepsilon$. Figure 12.12 depicts the schedule obtained by the strong stability algorithm at on-line execution.

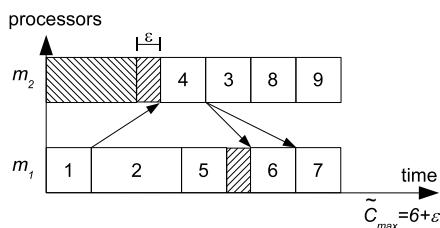


Figure 12.12. Schedule obtained by the strong stability algorithm after the stabilization

Let us note that the condition of strong stabilization is interesting from the theoretical point of view. However, it would be necessary to adapt it to weaker

conditions, thus limiting the disturbance magnitudes or accepting worst schedules produced by the disturbed static algorithm for small disturbances, for instance, by allowing more flexibility while adding pseudo-constraints.

In this part of the chapter, we have only considered small communication delays. In the following section we will consider large communication delays.

12.6. An intrinsically stable algorithm

In the first part of the chapter, we presented the stabilization approach. Let us recall again that the aim of stabilization is to reduce the disturbance impact on the performance of schedules computed at compile time. Basically, it consists of adjusting the schedule at run-time once the actual data are known. From the theoretical point of view, the stabilization approach can be applied on any schedule. However, in practice the experience shows that the effectiveness of the stabilization depends on the structure of the initial schedule. For instance, the stabilization has no significant effects on the schedule obtained by the linear version of DSC (this will be discussed in section 12.7.2). However, in this section we show that it is possible to design and develop a class of schedules based on clustering approaches that is naturally stable. This means it is able to absorb the bad effect of perturbations without using the stabilization process. Before establishing the result for intrinsic stability, we present this class of schedules and we briefly analyze its main interesting properties.

12.6.1. Convex clustering

In the context of parallel computing, much effort has been devoted to developing efficient static scheduling algorithms with low complexity. The most widely used methods are the *list scheduling* algorithms and the *clustering* based scheduling algorithms [SIN 07] (ETF and DSC are respectively two examples of such algorithms). We concentrate on the scheduling algorithms based on clustering. The clustering algorithms are in general for an unbounded number of processors. Clustering is an efficient way to reduce communication delay in the precedence task graph by grouping heavily communicating tasks to the same labeled clusters, then assigning tasks in a cluster to the same processor. In general, clustering algorithms have two phases: the task clustering phase that partitions the original task graph into clusters, and a post-clustering phase to merge the (folding) clusters generated in the previous phase onto a bounded number of processors and to order the task executions within each processor. We are interested here in the task clustering phase.

Lepère and Trystram in [LEP 02] proposed a clustering based on a structural criterion: to assign tasks to a location in *convex clusters*. Informally, a cluster is convex if, for any path whose extremities are in the same cluster, all intermediate tasks are also in that cluster. Figure 12.13(a) gives an example of convex clusters. On the contrary, Figure 12.13(b) shows an example of clusters that are not convex. In this case, the cluster containing tasks 1, 3, 6 is not convex because task 2 does not belong to it.

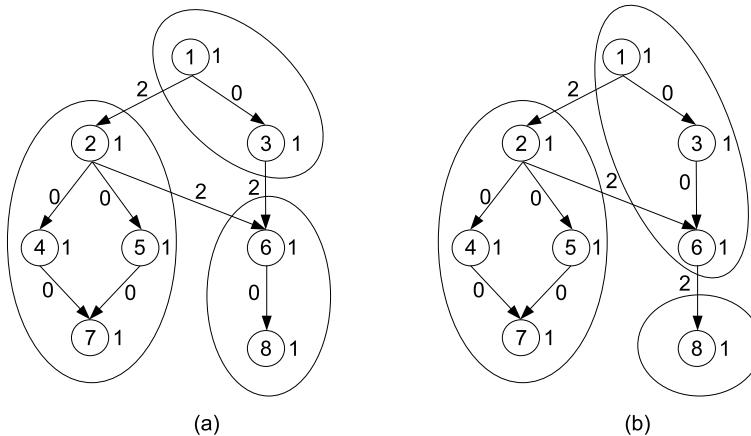


Figure 12.13. a) The graph is clustered on convex clusters;
b) we have a clustered graph which has no convex clusters

In the following, we present a class of cluster schedules that is intrinsically stable. We start with some definitions. Our notation is slightly different from that of [LEP 02]. From now on, let us consider a precedence task graph $G = (V, E)$ defined as earlier in the chapter. Let \prec be the partial order of the tasks in G and \sim the equivalence relation defined as follows: for x and $y \in V$, $x \sim y$ if and only if $x \not\prec y$ and $y \not\prec x$. If two tasks x and y are such that $x \sim y$ we say they are independent.

DEFINITION 12.2.—A clustering $R = \{V_i, \prec_i\}_i$ is a partition of the graph into groups of tasks associated with a total linear order extension of the original partial order \prec .

In practice, the groups V_i are composed of tasks that will be executed sequentially on the same processor. From each clustering, we can build a graph whose nodes are the clusters by adding all edges corresponding to the sequential execution inside the clusters. We call this graph *an induced graph* and we denote it by G_R^{induced} . All communication costs between tasks inside a same cluster are set to zero.

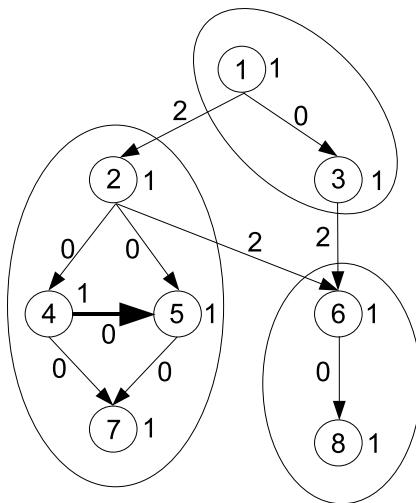


Figure 12.14. The induced graph $G_R^{induced}$ related to the clustering $R = \{\{1 \prec_1 3\}, \{2 \prec_2 4 \prec_2 5 \prec_2 7\}, \{6 \prec_3 8\}\}$ of Figure 12.13(a). The bold edge added between tasks 4 and 5 represents the sequential execution of tasks

If $R = \{V_i, \prec_i\}_i$ is a clustering, we derive a *cluster-graph* denoted by $G_R^{cluster}$ and defined as follows: the nodes of $G_R^{cluster}$ are the clusters of R , that is, the set of tasks $\{V_i\}_i$. There exists an edge between two distinct nodes V_i and $V_j \neq V_i$ if and only if there exist two tasks $x \in V_i$ and $y \in V_j$ such that $(x, y) \in E$. Figure 12.15 depicts an example of the cluster graph derived from the clustering of Figure 12.13(a).

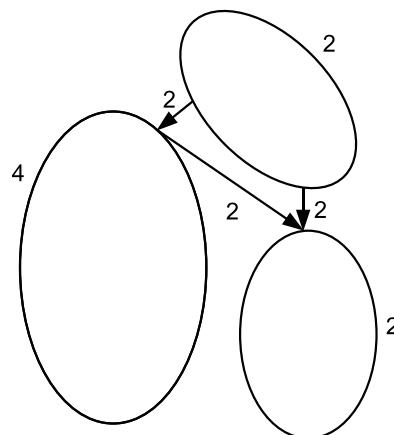


Figure 12.15. The cluster-graph $G_R^{cluster}$ related to the clustering depicted in Figure 12.13(a)

DEFINITION 12.3.– (Convex clustering) A clustering $R = \{V_i, \prec_i\}_i$ is a convex, if and only if the cluster-graph is acyclic.

The longest path in the graph G_R^{cluster} verifies the following lemma:

LEMMA 12.1.– For a convex clustering R , the longest path length of the graph G_R^{cluster} is an upper bound of its completion time C_{\max}^R .

The proof of lemma 12.1 is not developed here, so we refer the reader to [LEP 02]. Thus, the problem is to find a clustering $R = \{V_i, \prec_i\}_i$ of the graph G minimizing C_{\max}^R .

It is also possible to characterize the clusters of a convex clustering [LEP 02].

PROPERTY 12.2.– Every cluster of a convex clustering is a convex cluster.

Let us comment that this condition is insufficient since convex clusters do not imply systematically a convex clustering. We can now announce the main result of convex clusterings. Let C_{\max}^* be the completion time of an optimal clustering. Considering unit execution times and large communication delays the convex clustering are 2-dominants [LEP 02].

PROPERTY 12.3.– There exists a convex clustering R , such that $C_{\max}^R \leq 2C_{\max}^*$.

Proposition 12.3, whose proof is not detailed here, means that there exists a convex clustering whose execution time on an unbounded number of processors is no more than a factor of 2 from the optimal clustering. This property of 2-dominants of the convex clusterings justifies the fact of being only interested in the convex clusterings because from an approximation algorithm guaranteed for this problem of decomposition it is possible to deduce one performance guarantee for the problem of scheduling on an unbounded number of processors.

Another interesting property of convex clusterings is that the assignment obtained by gathering tasks in convex clusterings is *processor-ordered* (discussed in Chapter 4), that is, all the effective communications between two clusters occur from one cluster to the other cluster, and there are no communications in the opposite direction.

After having stressed the strength of convex clusterings it remains for us to find a “good” convex clustering. Several heuristics have been proposed for this problem, most of them based on a recursive decomposition of the precedence task graph, see for example [LEP 02] and [PEC 08]. For our purposes, in order to generate convex

clusterings we consider only the recursive decomposition proposed in [LEP 02]. The algorithm is called the *recursive convex clustering algorithm* (RCCA). We recall the principle of the RCCA as follows: first, the RCCA looks for the coarse grain parallelism. The algorithm determines four convex clusters of tasks A , \tilde{A} , $A^<$ and $A^>$, such that A and \tilde{A} are independent, i.e., can be executed simultaneously without communications, and such that the smallest of both groups is as large as possible. If such a decomposition leads to a clustering whose completion time is smaller than a sequential execution, then the RCCA applies the same scheme recursively on each cluster in order to find a smaller grain parallelism (fine grain). The sets of tasks $A^<$ and $A^>$ represent the sets of predecessors and successors, respectively, of both independent clusters A and \tilde{A} .

We illustrate the execution of the RCCA in an example. Figure 12.16 shows a precedence task graph composed of 16 tasks of unit processing time. The communication delays are constant (i.e., $c_{ij} = \rho$) and equal to $\rho = 2$. At a first stage, the graph is decomposed into four clusters $A^<$, A , \tilde{A} and $A^>$. On the one hand, by construction $C_{\max}^R \leq |A^<| + \rho + \max(|A|, |\tilde{A}|) + \rho + |A^>|$. In the example, at this stage $C_{\max}^R = 14$. The inequality is strict if one of the groups $A^<$ or $A^>$ is empty. On the other hand, $\max(|A|, |\tilde{A}|) + \min(|A|, |\tilde{A}|) = |A| + |\tilde{A}|$. Thus, $C_{\max}^R \leq |V| + 2\rho - \min(|A|, |\tilde{A}|)$. Moreover, from lemma 12.1 the clustering is interesting if $\min(|A|, |\tilde{A}|) > 2\rho$. If the communication delay ρ is small enough, the execution time of the clustering R is smaller than the sequential time. In this case, the RCCA is applied recursively on each group of the four induced graphs. Continuing with the example, only the graph induced by \tilde{A} can be decomposed again. After the decomposition of group \tilde{A} , the execution time of the clustering is 6 which is less than the time of a sequential execution of this cluster (in this example, the sequential execution of cluster \tilde{A} is equal to 7). Finally, the resulting clustering is $\{A^<, A, B, \tilde{B}, B^>, A^>\}$, whose execution time is equal to 12.

12.6.2. Stability analysis of convex clustering

Now, we study the impact of disturbances while scheduling convex clusterings. We show that any algorithm based on convex clusterings is stable since there is no cumulative effects of disturbances, that is, the intrinsic stability of convex clusterings. Based on this result we show that the schedule obtained by the RCCA is also intrinsically stable.

Since G_R^{cluster} is acyclic, if effective communications occur between two processors m_i and m_j , there are no communications in the opposite direction (i.e.,

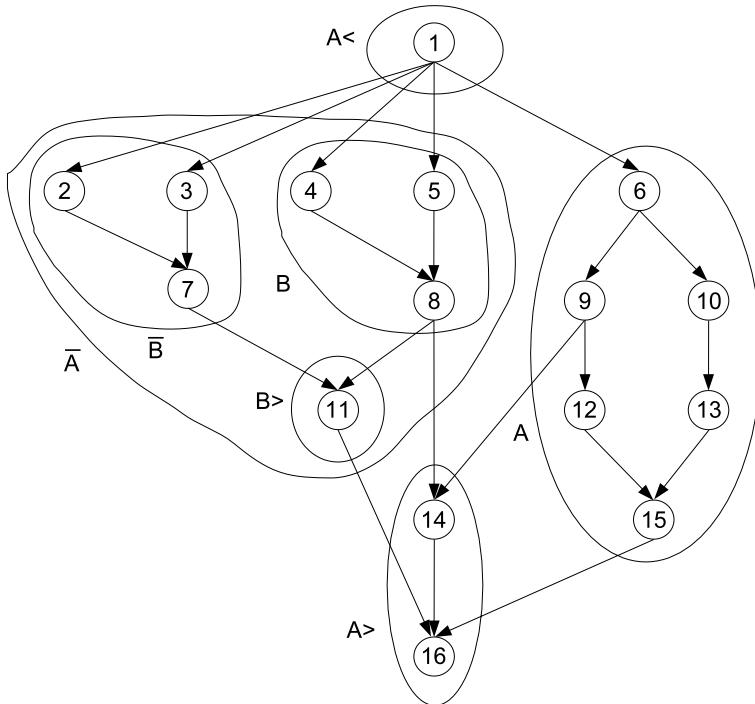


Figure 12.16. Recursive decomposition of a graph composed of 16 tasks.
The completion time after clustering is $C_{\max}^R = 12$

between m_j and m_i). Hence, the effects of disturbances are minimized. More specifically, if a task receives two perturbed communications originating from tasks allocated on different processors, its completion time is affected by only the maximum of the received disturbances.

Let us consider a convex clustering $R = \{(V_i, \prec_i)\}_i$ obtained by any decomposition heuristic and σ the computed schedule associated with this clustering. The tasks of each group are executed on a single processor. The problem of choosing the local policy (i.e., the total order on each group) of scheduling on each processor is not addressed here. It is important for practical implementations; however, let us emphasize that the theoretical analysis is valid for any local policy. Consider a static execution of this schedule, that is, an execution without changing the order of the task execution on the different processors.

Let us consider a task $u \in \{V_i\}$. $S(u)$ denotes the set of tasks in cluster V_i scheduled before task u , and task u included. For any task u , we note by

$P_c(u) = \{v \in \text{PREC}(u); \pi(u) \neq \pi(v)\}$ the set of communicating predecessors of task u (i.e., the predecessors of task u allocated on a different processor of that of task u). Finally, for any pair of communicating tasks v and u , we note by ϵ_{vu} the disturbance value of the communication c_{vu} .

Let us now consider a task t in a cluster $\{V_i\}$. Let C_t be its completion time on the schedule σ and \tilde{C}_t its actual completion time (i.e., after disturbance). We state the following property:

PROPERTY 12.4.–

$$\tilde{C}_t \leq C_t + \max_{v \in P_c(u), u \in S(u)} \epsilon_{vu}$$

Property 12.4 expresses that the maximum delay of a task t corresponds to the maximum disturbance over all the communications received by the tasks scheduled before task t , task t included. Suppose now that t is the last scheduled task in the cluster V_i , thus the maximum delay of t is equal to the maximum disturbances over all the communications received by all the tasks in this cluster.

Let us now consider the schedule σ . Let x be the task such that $C_x = C_{\max}$ and f be the tasks such that $\tilde{C}_{\max} = \tilde{C}_f$. We indicate by $\tilde{\mathcal{L}}$ the critical path in the disturbed graph. Applying property 12.4 recursively over the tasks in the critical path, the maximum delay of the completion time of task f is the sum of the disturbances along the path $\tilde{\mathcal{L}}$, that is, the number of communications along this path being noted $|\tilde{\mathcal{L}}|$, by indicating by ϵ_{\max} the maximal value of all the disturbances, we then obtain:

$$\begin{aligned} \tilde{C}_f &\leq C_f + |\tilde{\mathcal{L}}| \epsilon_{\max} \\ &\leq C_x + |\tilde{\mathcal{L}}| \epsilon_{\max} \end{aligned}$$

and thus the disturbed schedule length of a clustering computed by any heuristic or convex decomposition is bounded by:

THEOREM 12.1.–

$$\tilde{C}_{\max} \leq C_{\max} + |\tilde{\mathcal{L}}| \epsilon_{\max}$$

$|\tilde{\mathcal{L}}|$ being the number of edges along the path $\tilde{\mathcal{L}}$ means that it is the longest path (in number of edges) in the graph G_R^{cluster} .

We now turn to the recursive decomposition obtained by the RCCA. Based on its principle, we state below an upper bound on the number of edges in the longest path of $G_R^{cluster}$, which slightly improves the previous result and shows that any schedule obtained by the RCCA after disturbances in communication delay is intrinsically stable. Let $R = \{V_{i,j}, \prec_{i,j}\}_i$ be a convex clustering obtained by the RCCA and let m_p be the number of clusters in its longest path after p partitions.

PROPERTY 12.5.– *The maximum number of communications $|\tilde{\mathcal{L}}|$ in the longest path of $G_R^{cluster}$ is bounded by:*

$$|\tilde{\mathcal{L}}| \leq \frac{2}{3}(m_p - 1)$$

Thus, from theorem 12.1 and property 12.5:

COROLLARY 12.1.–

$$\tilde{C}_{\max} \leq C_{\max} + \frac{2}{3}(m - 1)\varepsilon_{\max}$$

Finally, as a concluding comment, let us emphasize that theorem 12.1 means that the degradation of the makespan computed without disturbances remains bounded. It achieves the stability of the initial schedule without needing on-line adaptation, and the bound is tight if and only if $\varepsilon_{ij} = \varepsilon_{\max}; \forall(i, j) \in E$, which is rare in practice.

12.7. Experiments

We report in this section some results and comparisons based on simulations of the schedules obtained by convex clusterings and those obtained by ETF and DSC after disturbances.

In order to generate convex clusterings, we have implemented the RCCA proposed in [LEP 02]. The two independent groups have been randomly selected. This process is repeated K times and we keep the best result. K is a constant that has been fixed experimentally to 10. We have implemented a simple local policy to schedule the tasks inside each cluster. The local policy is based on communicating tasks, that is, if there exists two tasks available at the same time, the priority is given to that with the largest number of successors, breaking ties randomly.

Next two stopping criteria of recursive decomposition were implemented. The first criterion was chosen on the cardinality of each cluster, that is, if the number of tasks inside each cluster is less than a constant, then stop the recursive decomposition. The constant is equal to 40 and is fixed experimentally. The second criterion is as follows. If a parallel execution of the cluster is more useful than a sequential execution, then apply the recursive decomposition, otherwise stop the decomposition. The choice of one criterion or another depends on the communication delays. The first criterion is for small communication delays and the second criterion for large communication delays.

We have used a linear version of DSC, that is, the clusterings obtained by DSC are linear. We recall that the RCCA and DSC consider an unbounded number of processors while ETF is designed for scheduling on a bounded number of processors. Thus, to compare the performances between the three algorithms, we first apply the RCCA and DSC to determine the number of processors (clusters), then ETF takes as input the smallest number of processors used by other two approaches to compute a solution.

We have generated random graphs using a Winkler graph generator [WIN 85]. This generator takes as parameter the size of the graph that will be generated. Winkler graphs are random graphs representative of multidimensional orders. To generate 2-dimensional order with n elements, n points are chosen randomly in the $[0..1] \times [0..1]$ square. Each point becomes a node and there is an edge between two points a and b if b is greater than a according to both dimensions. This kind of graph has only one task without predecessors.

12.7.1. Impact of disturbances in the schedules of the three algorithms

For this experiment we have generated a subset of random graphs from 40 to 1,340 tasks in size, with an increment of 10. We have generated 30 graphs for each measure point and we compare the average. For all the generated graphs we have assigned the parameters as follows: the processing time was randomly assigned from a uniform distribution in the interval $p_j \in [1, 20]$ and communication delays were randomly assigned in the interval $c_{ij} \in [1, 10]$. The disturbances concern the communication delays. We have considered that $\tilde{c}_{ij} \in [c_{ij}, 2c_{ij}]$, so the values have been uniformly chosen from this interval.

We compare the *stability ratio* (we refer the interested reader to the introduction of this book for more information) of the schedules produced for each algorithm without considering the stabilization process. It is defined as the ratio of the actual makespan (\tilde{C}_{\max}) and the initial makespan obtained at static execution (C_{\max}).

The stability ratio assesses the impact of disturbances in the schedules produced by the scheduling algorithms. A ratio close to 1 indicates a slight impact of the disturbances on the schedule. Conversely, any increase of this ratio implies that the schedule is greatly affected by the disturbances. Average ratio curves for the various scheduling algorithms are given in Figure 12.17. It can be seen from Figure 12.17 that the RCCA is more stable than DSC and ETF and the result is consistent while varying the size of the graph. We can observe that the RCCA ratio is close to one. This means that the makespan of the actual instance remains close to the initial makespan estimated without perturbation. We also observe that DSC is more stable than ETF because DSC is not sensitive to perturbations for the special configuration, that is, for applications with small communication delays and small disturbances [GER 95].

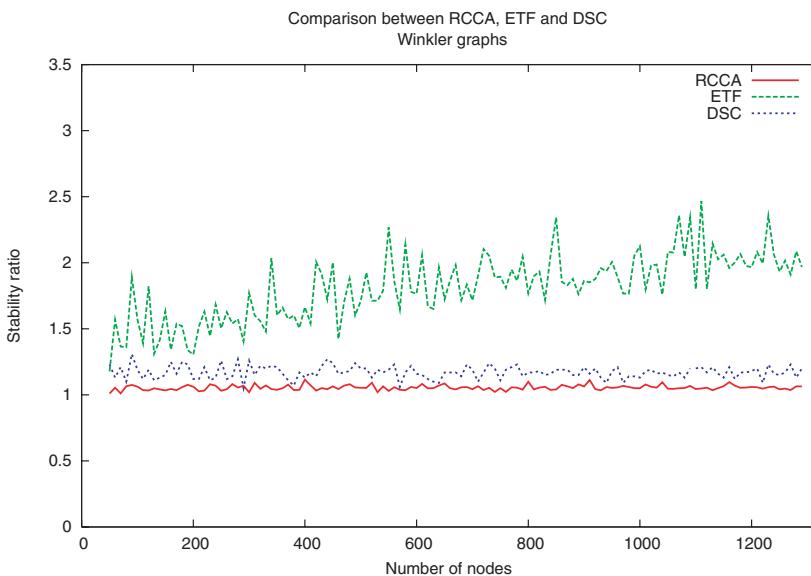


Figure 12.17. Performance comparison between RCCA, ETF and DSC after disturbances (small communication delays)

12.7.2. Influence of the initial schedule in the stabilization process

We now discuss the influence of the initial scheduling algorithm in the stabilization process by some experiments obtained from simulations in the case of small communication delays [MAH 03]. We stabilize the execution from the allocation calculated by the initial algorithm through the addition of pseudo-constraints. We give below a curve to estimate the efficiency of the strong stabilization algorithm

experimentally. In order to evaluate the influence of the initial schedule in the performance of the strong stabilization algorithm, we need only compare ETF and the linear version of DSC.

We used a performance criterion which is a slightly different definition of the *stability ratio*: it is defined as the ratio of the actual makespan (\tilde{C}_{\max}) and the makespan obtained after the on-line stabilization (\tilde{C}'_{\max}). This performance criterion assesses the improvement of an on-line execution with the stabilization process. Clearly, a ratio equal to 1 indicates that the actual makespan and the makespan obtained at on-line execution after the stabilization process are identical. On the contrary, a ratio greater than 1 shows an improvement in the actual makespan after the stabilization process.

We have generated another subset of Winkler graphs. The size of the graphs varies from 10 to 500 tasks, with increments of 10. We compare the average for 30 graphs generated for each measure point. The parameter values were assigned as in the previous experiment.

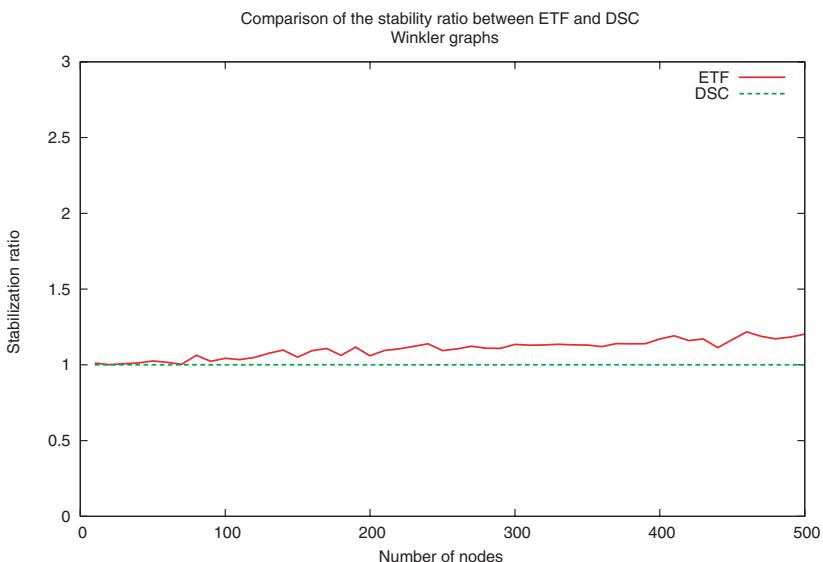


Figure 12.18. Stabilization ratio of ETF and DSC (small communication delays)

Average curves of the defined performance criterion of both algorithms are given in Figure 12.18. It can be observed that the ratio of the initial schedule computed by DSC is constant and equal to 1. This means that the schedule computed at static

execution after disturbances is equal to that computed at on-line execution after the stabilization process. Hence, the stabilization process has no effect on the schedules produced by DSC. This result is caused by the special structure of the schedule computed by DSC, that is, the version of DSC used gives linear clusterings and there are no independent tasks on an identical cluster. Thus, we cannot add pseudo-edges and no task permutations are possible. Hence, the task execution order at static execution is the same as the on-line execution leading to schedules with equal makespan. On the contrary, for an initial schedule obtained by ETF it is possible to add pseudo-edges avoiding bad permutations between tasks if some disturbances occur at on-line execution. In this case, the curve corresponding to ETF indicates an improvement on the makespan after the disturbances.

12.7.3. Comparison of the schedules with and without stabilization

Now, we compare the performance of the schedules with and without the stabilization process. We have generated random graphs with two different generators. The first is the Winkler generator and the second is a random layer graph generator [YAN 94]. This kind of generator takes as input three parameters, the size of the graph that will be generated, a parameter L which represents the number of layers composing the graph and a parameter pr which indicates the probability that two tasks are linked by an edge. We first generate the number of layers in each precedence task graph. Then, the tasks are fairly distributed in the layers. Next, we link the edges between tasks at different layers with a probability pr .

12.7.4. Test 1 – comparison for Winkler graphs

We compare the makespan of the RCCA with disturbances and ETF with and without the stabilization process for the case of large communication delays and small disturbances. For this experiment we have generated another subset of Winkler graphs. This subset contains graphs with 1000 to 3500 task nodes with increments of 20. For each measure point we have generated 30 graphs and we compare the average for each measure point. The processing time for the tasks was randomly assigned from a uniform distribution in the interval $p_j \in [1, 10]$ and the communication delays were randomly assigned in the interval $c_{ij} \in [10, 20]$. For this configuration, the RCCA used the second criterion to stop the recursive decomposition. For the disturbances, we have considered the same interval as in the previous experiment.

Figure 12.19 shows the results of comparing the RCCA against ETF with and without stabilization. As can be observed the schedules produced by the RCCA with

disturbances and ETF with stabilization are comparable. However, the stability of RCCA is slightly better than ETF with stabilization. On the contrary, the schedules computed by ETF at static execution are highly affected by the disturbances. It can be seen that ETF without stabilization gives worse schedules than the RCCA and ETF with stabilization. These results show that the RCCA is intrinsically stable without need of a stabilization process, that is, the schedules produced by the RCCA are able to absorb the bad effects of disturbances occurring at on-line execution. This is because of the special structure of the schedules produced by the algorithm.

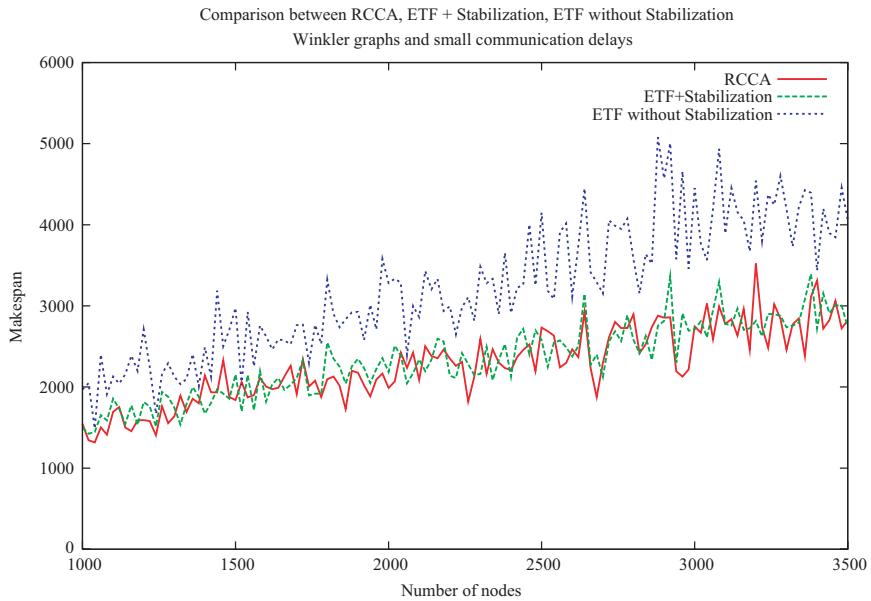


Figure 12.19. Performance comparison between RCCA with disturbances, ETF plus the stabilization process and ETF without the stabilization process (large communication delays)

12.7.5. Test 2 – comparison for layer graphs

The second set of graphs was generated using a random layer graph generator [YAN 94]. The size of the graph has been varied from 10 to 400 with an increment of 10. Again, for each measure point we have generated 30 graphs and we compare the average ratio for each measure point. For the values of processing time and communication delays we have used the same configuration as in the experiments of section 12.7.1. The disturbances have been uniformly chosen in the interval $\tilde{c}_{ij} \in [c_{ij}, 4c_{ij}]$.

Figure 12.20 depicts the results for the set of layer graphs. It can be seen that the stabilization process at on-line execution for the schedules obtained by ETF outperforms DSC with disturbances. That is, for this kind of graph the schedule computed by ETF is efficiently stabilized at on-line execution by the stabilization process, despite the fact that for applications with small communication delays without disturbances DSC outperforms ETF.

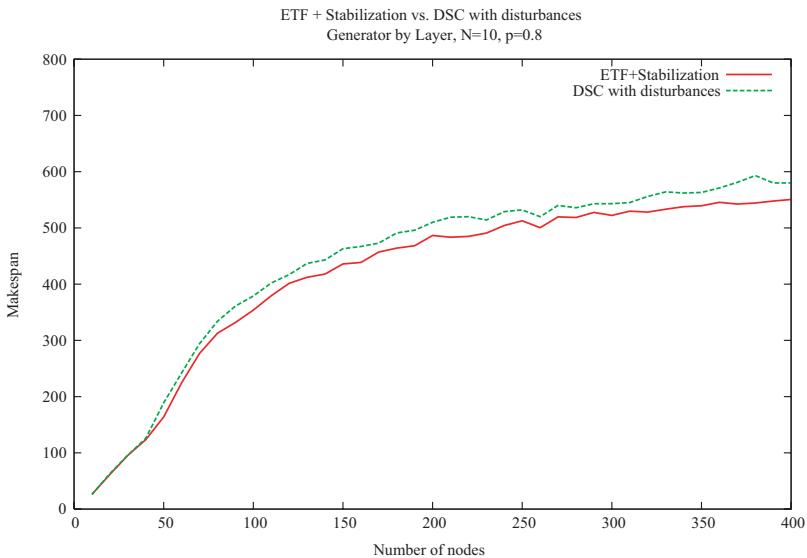


Figure 12.20. Performance comparison between ETF with stabilization and DSC after disturbances (layer graphs)

The efficiency of the stabilization process is dependent on the structure of the initial scheduling. The choice of such an algorithm is not often clear, indeed for instances with small and average communication delays DSC gives better results than ETF without taking into account disturbances [YAN 94]. On the contrary, if disturbances (large disturbances) occur at run time, then the schedule obtained by ETF could be efficiently improved by the stabilization process at on-line execution which is not the case of the schedule computed by DSC [MAH 03].

12.8. Conclusion

We have presented in this chapter an approach to deal with the disturbances appearing at run-time on communications for new parallel computing platforms. First, we have presented the stabilization process that aims at minimizing the effects of disturbances occurring during execution. Such a process guarantees that the

solution obtained at run-time on the disturbed instance is not too far from the solution computed on the initial instance. Then, we have described a scheduling algorithm which is intrinsically stable.

However, the stabilization approach is interesting and efficient only when the disturbances are not too important, i.e. when the actual instance remains close to the initial one. In practice, the differences might be important, and in that case, it is more useful to use a purely on-line approach.

Acknowledgments

We would like to thank Eric Sanlaville for his helpful comments and suggestions.

12.9. Bibliography

- [BUY 99] BUYYA R. (Ed.), *High Performance Cluster Computing: Architectures and Systems*, vol. 1, Prentice Hall, NJ, USA, 1999.
- [CUL 99] CULLER D.E., SINGH J.P. and GUPTA A., *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kauffman Publishers, San Francisco, CA, 1999.
- [FOS 99] FOSTER I. and KESSELMAN C. (Eds.), *The Grid: Blueprint for a New Computing Infrastructure*, vol. 1, Morgan Kauffman Publishers, Inc., San Francisco, USA, 1999.
- [GER 95] GERASOULIS A. and YANG T., “Applications of graph scheduling techniques in parallelizing irregular scientific computation”, in FERREIRA A. and ROLIM J. (Eds.), *Parallel Algorithms for Irregular Problems: State of the Art*, Chapter 13, p. 245–267, Kluwer Academic Publishers, 1995.
- [GER 01] GERMAIN C., FEDAK G., NERI V. and CAPPELLO F., “Global computing systems”, *3rd International Conference on Scale Scientific Computations*, Lecture Notes in Computer Science; vol. 2179, p. 218–227, 2001.
- [GRA 66] GRAHAM R.L., “Bounds for certain multiprocessing anomalies”, *Bell System Technical Journal*, vol. 45, p. 1563–1581, 1966.
- [GUP 03] GUPTA A., PARMENTIER G., TRYSTRAM D., “Scheduling precedence task graph with disturbances”, *RAIRO Operations Research*, vol. 37, p. 145–156, July 2003.
- [HWA 89] HWANG J.J., CHOW Y.CH., ANGERS F.D. and LEE CH.Y., “Scheduling precedence graphs in systems with interprocessor communication times”, *SIAM Journal on Computing*, vol. 18, no. 2, p. 244–257, April 1989.
- [LEP 02] LEPÈRE R., TRYSTRAM D., “A new clustering algorithm for scheduling with large communication delays”, *16th IEEE-ACM Annual International Parallel, Distributed Processing Symposium (IPDPS'02)*, Fort Lauderdale, USA, April 2002.

- [LEU 04] LEUNG J.Y-T. (Ed.), *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Chapman & Hall/CRC, Boca Raton, FL, USA, 2004.
- [MAH 03] MAHJOUB A., RAPINE C., TRYSTRAM D., “Influence of starting solutions on the stabilization of scheduling algorithms”, *Euro-Informs*, Istanbul, Turkey, July 2003.
- [MAN 67] MANACHER, G.K., “Production and stabilization of real-time task schedules”, *Journal of the ACM*, vol. 14, no. 3, p. 439–465, July 1967.
- [MOU 99] MOUKRIM A., SANLAVILLE E. and GUINAND F., “Scheduling with communication delays and on-line disturbances”, in *Euro-Par’99 Proceedings of the 5th International Euro-Par Conference on Parallel Processing*, Toulouse, France, p. 350–357, 1999.
- [PAP 90] PAPADIMITRIOU C.H. and YANNAKAKIS M., “Towards an architecture-independent analysis of parallel algorithms”, *SIAM Journal of Computing*, vol. 19, no. 2, p. 322–328, 1990.
- [PEC 08] PECERO SÁNCHEZ J.E., Local-global scheduling interactions, PhD Thesis, Grenoble Institute of Technology, INPG, Grenoble, France, 2008.
- [PEN 01] PENZ B., RAPINE C. and TRYSTRAM D., “Sensitivity analysis of scheduling algorithms”, *EJOR*, vol. 134, p. 606–615, November 2001.
- [RAY 87] RAYWARD-SMITH V.J., “UET scheduling with unit interprocessor communication delays”, *Discrete Applied Mathematics*, vol. 18, p. 55–71, 1987.
- [SHM 95] SHMOYS D.B., WEIN J. and WILLIAMSON D.P., “Scheduling parallel machines on-line”, *SIAM Journal on Computing*, vol. 24, p. 1313–1331, December 1995.
- [SIN 07] SINNEN O., *Task Scheduling for Parallel Systems*, John Wiley & Sons, Hoboken, New Jersey, 2007.
- [ULL 75] ULLMAN J.D., “NP-complete scheduling problems”, *Journal of Computer and System Sciences*, vol. 10, no. 3, p. 384–393, 1975.
- [WIN 85] WINKLER P., “Random orders”, *Order*, vol. 1, p. 317–331, 1985.
- [YAN 92] YANG T. and GERASOULIS A., “PYRROS: static task scheduling and code generation for message passing multiprocessors”, in *Proceedings of the 6th ACM International Conference on Supercomputing*, Washington DC, p. 428–437, July 1992.
- [YAN 93] YANG T. and GERASOULIS A., “List scheduling with and without communication delay”, *Parallel Computing*, vol. 19, no. 12, p. 1321–1344, 1993.
- [YAN 94] YANG T. and GERASOULIS A., “DSC: scheduling parallel tasks on an unbounded number of processors”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, p. 951–967, September 1994.

Chapter 13

Contribution to a Proactive/Reactive Control of Time Critical Systems

13.1. Introduction

This chapter focuses on *time critical systems* (also called *time window constraints*) in which task processing times are given by time intervals instead of a single value. The control of such systems will be treated in a periodic functioning mode. Time is a crucial notion in these systems because a feasible behavior not only depends on the logical correctness of the task sequence, but also on the task completion times. Indeed, the system may enter into a forbidden state if a required result or a required event occurs too early or late.

These time intervals may correspond either to a temporal flexibility associated with the processing time of the task; or to an uncertainty as to its real duration due to the performing resource (machine or human); or also to take account of the deterioration of the production equipment.

Such systems can be found in computer systems where supervision applications require data consistency, in manufacturing systems of chemical industry where chemical reagents are efficient during a given time interval or in the food industry, where handling and delivery of products are subjected to freshness requirements.

Control is mandatory to guarantee the logical and temporal specification constraints of these systems. Because processing tasks lead to temporal constraint violations as soon as possible, it is necessary to solve the sequencing problem by considering uncertainty. Moreover, these systems constitute an attractive framework for the validation of a proactive/reactive control in a disturbed environment. A transgression of temporal specifications causes a decrease of the quality of service and such an approach is justified.

The proposed approach follows the three steps defined in section 1.3.3.1 of Chapter 1:

– *Step 0*: this step consists of modeling the considered system to bring out the set of temporal and logical constraints the schedule has to respect. In others words, the operations sequence and the required resources are known and modeled. The systems considered here are called “time critical systems”. A time intervals modeling tool is applied to translate a temporal tolerance on the task durations or an uncertainty on its effective duration. Directly formulating an analytical expression of the constraints in this context is difficult. Thus, Petri nets [BRA 82, DAV 92] are used as a modeling tool for this phase. More precisely, p-time Petri nets [KHA 97] are used because time intervals are associated with the sites of the Petri net. Their firing rules impose a firing compulsion whatever the considered activities are. As the model includes the process and the routing, it provides the set of temporal and logical constraints to be considered in step 1.

– *Step 1*: the second step aims at computing a set of feasible schedules. When the system is modeled by a Petri net, solving a static sequencing problem is determining the sequencing of tasks (a particular firing sequence is imposed, so that the model becomes deterministic) and setting the starting and completion times of each operation by computing their firing instants.

The investigation of all the feasible sequences is made by enumerative analysis that does not require any strong property of the net structure. Among the existing enumerative analysis approaches [BER 83, KHA 97, BOU 93, BON 01a], we use the one based on firing instants. This approach allows an absolute time referencing. Thus, enumeration is potentially infinite because the same states are never found again in regard of time evolution, but the definition of state classes avoids this problem [BOU 93, BON 01a]. Moreover, together with linear programming techniques, this approach makes the following possible:

- determination of the (minimum and maximum) performances of each cyclic repetitive enumerated functioning mode as well as the (minimum and maximum) duration of the transient functioning leading from the initial state;
- computation of a reference schedule (the firing transition instants of the chosen sequence). Criterion may be traditional (cycle time, makespan, etc.) or may refer to temporal slacks (supervision intervals) that will initialize the following step.

It is possible to evaluate each enumerated sequence and to select one of them. Generally, this selection needs a multicriterion approach (work-in-progress, cycle time, number of solutions in case of failure). The chosen sequence is the reference sequence for the next step.

– *Step 2*: this step introduces robustness and leads to a reactive dynamic control. The disturbances that are considered here are earliness or tardiness of the task starting times. The firing instances are updated and, depending on the level of flexibility, can lead to a sequence modification [AYG 03, BON 01b]. The criterion used for updating allows us to determine for all these transitions, all the time intervals in which the associated event must occur. Of course, this procedure has to be iterated on-line.

These steps are developed in the following sections. Each phase of the procedure is illustrated on a given manufacturing process.

13.2. Static problem definition

P-time Petri nets [BRA 82, MUR 89, DAV 92] offer an efficient and recognized tool for modeling discrete event systems, thanks to their graphical nature, their ability to model parallel and distributed processes and their firm mathematical foundation. A detailed presentation of this tool is made in the references above. Petri nets (or PNs) have been extended and modified in several ways to take into account the system functions and to represent explicitly the temporal specifications. Deterministic extensions are classified into two categories: the time PN [RAM 74, SIF 77] where time attributes are single values and the p-time PN [MER 74, VAN 92, DIA 94, KHA 97] where these ones are time intervals representing either feasibility gaps of the operations or uncertainties on the process. Due to the features of time critical systems, p-time PNs have been naturally chosen because they are able to easily represent synchronization in a constrained time: in other words, resources must be available in a time compatible with their exploitation. Notice that our aim is not to develop all the specificities of this model but only to give an overview allowing its use in a sequencing problem. Nevertheless, key definitions of autonomous PNs are quickly recalled in the following section.

13.2.1. Autonomous Petri nets (APN)

A PN is composed of two parts: one is static – the structural part – and one is dynamic – the behavioral part. The structural part is made of an oriented and valued bipartite graph denoted by $\langle P, T, \text{Pre}, \text{Post} \rangle$, where:

- P is a finite set of n places noted $\{p_1, p_2, \dots, p_n\}$,
- T is a finite set of m transitions noted $\{t_1, t_2, \dots, t_m\}$ with $P \cap T = \emptyset$,
- $\text{Pre}: P \times T \rightarrow \mathbb{N}$ is the input incidence application, corresponding to the direct arcs linking places to transitions,
- $\text{Post}: P \times T \rightarrow \mathbb{N}$ is the output incidence application, corresponding to the direct arcs linking *places to transitions*.

Using linear algebra, the input and output incidence matrices and the flow matrix are defined by:

- $\text{Pre} = [\delta_{ij}]$, ($1 \leq i \leq n$, $1 \leq j \leq m$) with $[\delta_{ij}] = \text{Pre}(p_i, t_j)$.
- $\text{Post} = [\epsilon_{ij}]$, ($1 \leq i \leq n$, $1 \leq j \leq m$) with $[\epsilon_{ij}] = \text{Post}(p_i, t_j)$.
- $W = [w_{ij}]$, ($1 \leq i \leq n$, $1 \leq j \leq m$), with $w_{ij} = \text{Post}(p_i, t_j) - \text{Pre}(p_i, t_j)$.

Let us denote by:

– ${}^\circ t$ (respectively t°) the set $\{p \in P / \text{Pre}(p, t) \neq 0\}$ (respectively $\{p \in P / \text{Post}(p, t) \neq 0\}$) of input places (respectively output places) of transition t .

– ${}^\circ p$ (respectively p°) the set $\{t \in T / \text{Post}(p, t) \neq 0\}$ (respectively $\{t \in T / \text{Pre}(p, t) \neq 0\}$) of input transitions (respectively output transitions) of place p .

An initial marking $M_0 : P \rightarrow \mathbb{N}$, where $M_0(p)$ indicates the number of tokens that are initially in place p is added to the static part. The dynamic part of the net is realized by the marking modification due to the firing of transitions.

DEFINITION 13.1.– A transition t is fireable for marking M if: $\forall p \in P$, $M(p) \geq \text{Pre}(p, t)$.

If the transition is fireable, it may be fired, leading to a new state: the marking M' obtained from marking M by the firing of t is given by the following *fundamental equation*:

$$\forall p \in P, \quad M'(p) = M(p) + \text{Post}(p, t) - \text{Pre}(p, t)$$

Let $\sigma = t_1 t_2 \cdots t_k$ be a firing sequence of transitions which can be performed from an initial marking M_0 . The fundamental equation gives the new marking M_k :

$$M_k = M_0 + W \times \underline{\sigma}$$

where $\underline{\sigma}$ is the characteristic vector of sequence σ (each of the m components is the number of transition firing occurrences t in sequence σ).

Notice that the fundamental equation is a necessary condition for *accessibility*. Thus, a positive vector verifying this equation is not necessarily a feasible sequence for the considered net.

13.2.2. *p*-time PNs

The formal definition of a p-time PNs [KHA 97] is given by a pair $\langle N, SI \rangle$ where:

- N is a marked PN.
- $SI: P \rightarrow \mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \{+\infty\})$ with \mathbb{Q}^+ the set of positive rational numbers.
 $p_i \rightarrow SI(p_i) = [a_i, b_i]$ with $a_i \leq b_i$.

$SI(p_i)$ defines the static time interval of a token in a place p_i and specifies the minimum and maximum “waiting” time in this place. The model is no longer autonomous: the state of a p-time PN refers, in addition to the marking, to the temporal state of the tokens in the sites.

DEFINITION 13.2.– At a given time, the state S of a p-time PN is totally defined by the pair (M, I) where:

- M is a vector of size n that contains the number of tokens in each place of the net, for the considered state ($\forall p \in P, M(p) \geq 0$),
- I is a “potential interval” application of firing that associates a time interval $[a_i^k, b_i^k]$ with each token k in place p_i . This is called a “dynamic interval”, to be distinguished from the static interval associated with the place containing this token. These intervals are related to the time the token arrives in the place.

Thus, if token k arrives at instant τ in place p_i then at instant $(\tau + d)$ (with $d \leq b_i$), the dynamic interval of token k is $[\max(0, a_i - d), b_i - d]$. Notice that:

- A new state may be reached by the passage of time and not only by transition firing, as in an autonomous model.

– Due to the continuity of time, generally, an infinite number of states may be reached from a given state.

– Two problems may occur in the p-time PN: the first one is the classical blocking where no transition is enabled. The second one is specific to the considered model: the dynamic interval of the token is $[0, 0]$ and no output transition of the place is enabled at this instant. Such a token is defined as a dead-token [KHA 96] indicating a forbidden state.

This model imposes control on the transition firing, to guarantee the potential constraints of the places. The firing conditions of a transition generate three states for a token, depending on the “waiting” time in the place:

– The state “not available”, when the waiting time is less than the temporal minimum bound of the place containing it. The significance of this situation is that the task is being performed. This token cannot be used for a transition firing.

– The available state, when its waiting time is found between the minimum and the maximum bound. In this case, the task may be achieved at any time. This token must be used by the firing of a transition (firing compulsion).

– The dead-token state, when its waiting time is greater than the maximum bound. This state represents the transgression of the temporal specifications of the model. The token remains in place and *can no longer* participate in the validation of the transitions.

Thus, for determining the firing interval of the enabled transition t_i , it is not sufficient to consider only the tokens in the set of the input places of t_i (${}^{\circ}t_i$). All the tokens of the net must be taken into account, even if they don't help enable transition. The two following definitions allow the expression of an accessibility relation between different states in a p-time PN.

DEFINITION 13.3. – State $E' = (M', I')$ is reached from state $E = (M, I)$ by the running of time τ if and only if: $M' = M$, $\forall j$, a token in place p_i , $a_i^{j'} = \max(a_i^j - \tau, 0)$ and $b_i^{j'} = b_i^j - \tau$, with $b_i^{j'} \geq 0$, where a_i^j and b_i^j (respectively $a_i^{j'}$ and $b_i^{j'}$) represent the minimum and maximum bounds of the dynamic interval associated with token j in place p_i , for state E (respectively E').

DEFINITION 13.4. – State $E' = (M', I')$ is reached from state $E = (M, I)$ by the transition firing t_i if and only if:

– t_i is fireable from E (i.e. the minimum bound of its dynamic interval is equal to 0),

$$-\forall p \in P, M'(p) = M(p) - \text{Pre}(p, t_i) + \text{Post}(p, t_i),$$

– the tokens that remain in places keep their dynamic intervals in E' (the firing time is considered as 0). The dynamic intervals of the new tokens created by the firing of t_i are initialized by the temporal interval associated with the places into which they are dropped.

Among the controlled functioning modes of p-time PNs, the 1-periodic mode (denoted by periodic mode in the sequel) is such that the time elapsed between two successive firings of transition t_i is always equal to the same quantity. This quantity defines the functioning period. The formal definition of this mode is given by the following theorem:

THEOREM 13.1 [RAM 74].– *The behavior of the periodic mode is fully determined by: $\forall k, k \geq 1, S_i(k) = S_i(1) + (k - 1) \times \pi$, where $S_i(k)$ is the k th firing instant of transition t_i and π the functioning period.*

The first firing instants of the transitions and the functioning period are sufficient to describe entirely the periodic functioning mode. As this mode provides the same performance as any other controlled functioning mode [CAL 97], this mode will be used in step 1 for building a reference schedule.

Illustrative example

To illustrate the presented methodology, a part of a graphite ceramic workshop is considered. The raw materials (primarily coke and binder pitch) are first mixed together at a high temperature in order to obtain a homogenous mass. The paste obtained is then conditioned (the paste is cooled to bring down its temperature). Finally, the paste is extruded into the shape and size of end-products. A time interval is associated with each operation. This interval represents an uncertainty on the duration (*for example, the duration of the conditioning depends on the temperature of the ambient air, of the paste and of the conditioner*). It may also correspond to the duration in which the good properties of the pitch are preserved (*transfer, extrusion, etc.*). The line is composed of six mixing tanks, two conveyors, two conditioners and one press.

The modeling of the workshop is completed in two phases. The first one is the modeling of the precedence (succession) constraints extracted from the manufacturing process (linear central part of the graph): places model the operations and transitions are the beginning and ending events.

Then, in the second step of the modeling, the resources necessary to perform these operations are added up. Thus, transition t_2 is synchronized in a constrained time:

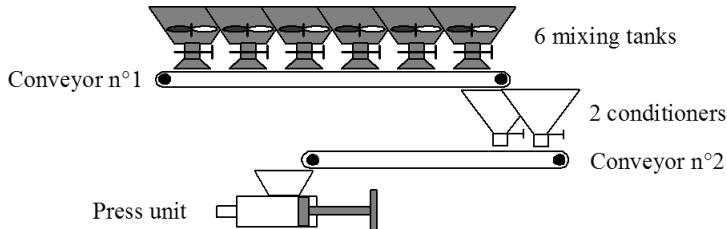
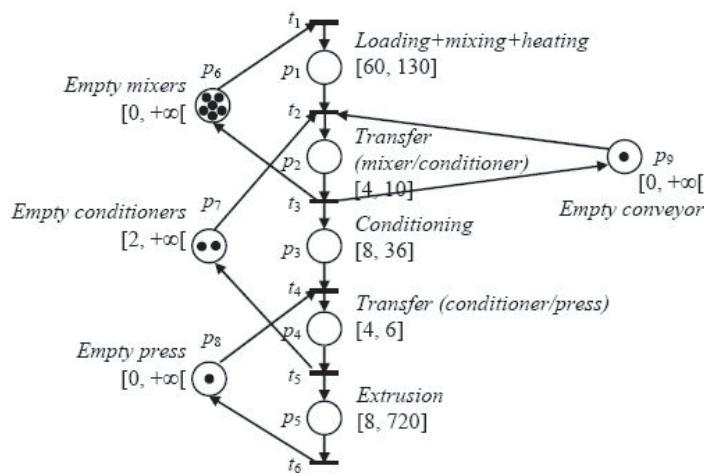


Figure 13.1. The graphite ceramic cell

Figure 13.2. p -time PN model of the cell

the conveyor and one of the conditioners must be free in a time compatible with the end of the mixing operation. The initial marking of the net represents the idle state in which the resources are available. To illustrate this, the number of tokens in $p_6(6)$ corresponds to the number of mixers, the number of tokens in $p_7(2)$ corresponds to the number of conditioners and number of tokens in $p_8(1)$ corresponds to the only available press. Notice that only the conditioners necessitate a reuse time (duration between two consecutive utilizations) which is a minimum of 2 time units. Thanks to this model, it is possible to generate all the constraints of the sequencing problem of this workshop. This is developed in the next section.

13.3. Step 1: computing a feasible sequencing family

Enumerative analysis, also identified as exhaustive simulation, is an efficient validation method for a p-time PN. It is used here to list all the feasible sequences of the p-time PN of the considered system. Relying on the firing rule of the transitions, it facilitates the building of a state class tree similar to the reachability graph obtained for autonomous PNs. Thus, if the obtained tree (graph) is finished, the behavior of the system can be investigated by analyzing some of its interesting properties: boundedness, liveness, existence of steady states, existence of live tokens sequences in the case of the p-time model, etc.

All the notions that contribute to this analysis are not be evoked here (for more details, see [BOU 93, BON 01a]). Only the relevant ones, used to build the set of constraints related to an enumerated sequencing, are developed.

The fundamental feature of the p-time PNs is the temporal state of the tokens. In the sequel, it will be considered that a token is identified by its place (static aspect) as well as its consumption instant (dynamic aspect). In the case of 1-valued arcs, this induces the following definition of the function TOK :

DEFINITION 13.5. – $TOK: \mathbb{N} \times \mathbb{N}^* \rightarrow \mathcal{P}(P)$, with $\mathcal{P}(P)$ the set of the parts of P .
 $TOK(j, n) = \{p \in P / \text{a token exists in place } p, \text{ created by the } j\text{th firing instant with}$
 $\text{and consumed by the } n\text{th, with } j < n\}$ $TOK(0, n) = \{p \in P / \text{a token exists at the}$
 $\text{initial instant in place } p, \text{ and consumed by the } n\text{th, with } n > 0\}$.

In the case of a multi-validation, for differentiating the tokens contained in the same place, index j and n are associated with them. Indeed, by using these indices, it is possible to have any type of management of the tokens in place. A FIFO management is considered here. Notice that the building of the sets TOK depends directly on the considered sequence (the firing sequence). From these sets $TOK(j, n)$, the minimum and maximum availability durations of the tokens in the places are related to the static time intervals associated with the considered places and their expression are:

$$\begin{aligned} Dsmin(j, n) &= \begin{cases} \max_{i/p_i \in TOK(j, n)} (a_i) & \text{if } TOK(j, n) \neq \emptyset \\ 0 & \text{if } TOK(j, n) = \emptyset \end{cases} \\ Dsmax(j, n) &= \begin{cases} \max_{i/p_i \in TOK(j, n)} (b_i) & \text{if } TOK(j, n) \neq \emptyset \\ 0 & \text{if } TOK(j, n) = \emptyset \end{cases} \end{aligned}$$

with a_i and b_i the bounds of the static interval $([a_i, b_i])$ of place p_i .

Enumerative analysis is based on the notion of firing instants introduced in [BOU 93]. For the sake of simplicity, the index of transitions are classified according to the sequence considered. Figure 13.3 illustrates the following σ sequence, $\sigma = t_1 t_2 \cdots t_q$.

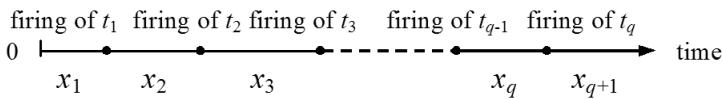


Figure 13.3. Firing instants

Thus, x_1 is the firing instant of transition t_1 , $(x_2 + x_3)$ is the time elapsed between the third firing instant and the first (i.e. the time elapsed between the firing of t_1 and the firing of t_3), etc. To simplify the expressions in what follows, x_i will represent the firing instant of the i th fired transition (t_i). Indeed, this is true if the reasoning is made relatively to the transition firing preceding this of t_i ; in absolute value, the firing instant of t_i corresponds to the sum $\sum_{k=1}^i x_k$.

First firing instant: a transition t_1 initially enabled will be the first fired if and only if it exists $x_1 \geq 0$ such that: $Dsmin(0, 1) \leq x_1 \leq Dsmax(0, i), \forall i, 1 \leq i \leq n$.

The left part expresses the availability of the tokens engaged in the firing of transition t_1 (i.e. only the tokens located in input places of the transition considered and participating to the firing are taken into account). The right part gives the set of the constraints preserving from death all the tokens present in the net at the evaluation instant. Thus, the consideration of the tokens located in the input places of t_1 is insufficient.

Note: in the expression $Dsmax(0, i), \forall i, 1 \leq i \leq n$, the quantity n represents only the fact that at the evaluation of the first firing instant, the set of the tokens initially present in the net are considered, without taking care over their consumption instant.

qth firing instant: a transition t_q enabled by the firing instant i ($0 \leq i \leq q - 1$) will be the q th to be fired (after the firing of t_1, t_2, \dots, t_{q-1}) if and only if x_q exists such that:

$$Dsmin(u, q)_{u \in SEN(q)} \leq \sum_{s=u+1}^q x_s \leq Dsmax(u, k)_{u \in SEN(q), k=q, \dots, n} \quad (13.1)$$

with $\text{SEN}(q) = \{u / \text{TOK}(u, q) \subset (^o t_q)\}$, and

$$\sum_{s=i+1}^q x_s \leq \min(\text{Dsmax}(j, k)_{\substack{j=0, \dots, i-1 \\ k=q+1, \dots, n}} - \sum_{s=j+1}^i x_s) \quad \text{with } i > 0 \quad (13.2)$$

and

$$\sum_{s=i+1}^q x_s \leq \min \left(\text{Dsmax}(j, k)_{\substack{j=i+1, \dots, q-1 \\ k=q+1, \dots, n}} + \sum_{s=i+1}^j x_s \right) \quad \text{with } i+1 < q \quad (13.3)$$

$\text{SEN}(q)$ represents the creation instants set of tokens consumed by the q th firing instant. For the lower bound, the set of inequalities (13.1) represents the contribution of the tokens participating to the firing considered (i.e. the q th, their creation instant being of course less than or equal to i). For the upper bound, the previous set is extended to the tokens which have a creation instant less than or equal to i , but with a consumption instant possibly greater than q . The set of inequalities (13.2) represents the influence of the tokens created after the enabling instant i and consumed after the q th firing instant. The set of inequalities (13.3) represents the impact of the tokens created after the i th firing instant and consumed after the q th. After the addition of $\sum_{s=0}^i x_s$ to the previous results, the following conditions are obtained:

$$\text{Dsmin}(i, q) + \sum_{s=0}^i x_s \leq \sum_{s=0}^q x_s \leq \min_{\substack{j=0, \dots, q-1 \\ k=q, \dots, n}} \left(\text{Dsmax}(j, k) + \sum_{s=0}^j x_s \right),$$

and

$$\text{Dsmin}(u, q)_{u \in \text{SEN}(q) \setminus \{i\}} \leq \sum_{s=u+1}^q x_s \leq \text{Dsmax}(u, k)_{u \in \text{SEN}(q) \setminus \{i\}; \substack{k=q, \dots, n}},$$

To express the obtained results more simply, the definition of the following coefficients is required:

$$c_{uq} = \begin{cases} \text{Dsmin}(u, q) & \text{if } u \in \text{SEN}(q) \\ 0 & \text{otherwise} \end{cases},$$

$$d_{jk} = \begin{cases} \text{Dsmax}(j, k) & \text{if } \text{TOK}(j, k) \neq \emptyset \\ +\infty & \text{otherwise} \end{cases}$$

with $\forall (j, k) \in [0, q-1] \times [1, q]$, $j \notin \text{SEN}(q)$ and $k \neq q$, then $c_{jk} = 0$, and $\forall k \in [0, q]$, $x_k \geq 0$.

Intuitively, notice that c_{uj} values are referred to as a set of tokens enabling a particular transition and initiating its firing. That is why we specify that the c_{uj} values associated with the tokens that do not participate in the q th firing of the transitions are inhibited. The d_{jk} values refer to the tokens present in the net at the observation instant.

The following condition is finally obtained:

$$\max_{\substack{j=0, \dots, q-1 \\ k=q, \dots, n}} \left(c_{jk} + \sum_{s=0}^j x_s \right) \leq \sum_{s=0}^q x_s \leq \min_{\substack{j=0, \dots, q-1 \\ k=q, \dots, n}} \left(d_{jk} + \sum_{s=0}^j x_s \right)$$

This result can be generalized to a firing sequence.

DEFINITION 13.6. – A sequence of transitions $t_1 t_2 \dots t_q$ may be fired respectively at firing instants $1, 2, \dots, q$ if and only if $x_1 \geq 0, x_2 \geq 0, \dots, x_q \geq 0$ exist such that:

$$S_\sigma(q) = \begin{cases} c_{0k} \leq x_1 \leq d_{0k}, k = 1, \dots, n \\ \max_{k=2, \dots, n} (c_{0k}, c_{1k} + x_1) \leq x_1 + x_2 \leq \min_{k=2, \dots, n} (d_{0k}, d_{1k} + x_1) \\ \vdots \\ \max_{\substack{j=0, \dots, q-1 \\ k=q, \dots, n}} \left(c_{jk} + \sum_{s=0}^j x_s \right) \leq \sum_{s=0}^q x_s \leq \min_{\substack{j=0, \dots, q-1 \\ k=q, \dots, n}} \left(d_{jk} + \sum_{s=0}^j x_s \right) \end{cases}$$

DEFINITION 13.7. – The firing space at the q th firing instant denoted by $\text{FSP}(q)$ is the set of non-negative vectors (x_1, \dots, x_q) such that the first, the second, etc. and the q th firing conditions are satisfied.

In a cyclic functioning mode, the repetitive firing sequence associated with the p-PN makes it possible to deduce easily from the model the suitable values $\text{TOK}(\cdot)$. Thus, from these values, the inequalities system representing the firing space associated with the repetitive sequence can be expressed.

The assumption made on the initial marking (available or not for the firing of the initially enabled transitions) identifies the system as relaxed or non-relaxed. Whatever the case, the knowledge of a repetitive firing sequence is not sufficient to determine the bounds of its performance evaluation. Thus, the existence of initially present tokens amounts to considering that the steady state is reached as soon as they leave the system (*the FIFO management of the tokens in the different places is significant*).

To face this particularity, it is necessary to build the firing space associated with the considered repetitive sequence until index q , with q determined as follows.

Let us consider a firing finite sequence σ , with $\sigma = t_1 t_2 \dots t_s$ (*the transitions are indexed according to their firing order in σ*). Let us denote by:

- quantity $|\sigma|$ the number of transitions composing the sequence σ (here, $|\sigma| = s$),
- mapping $ord_\sigma: T \rightarrow N^*$, $t_u \rightarrow ord_\sigma(t_u)$ the function representing the occurrence of t_u in the sequence σ .

So, for each transition t_u of the net, quantity n_{tu} is evaluated by:

$$n_{tu} = \max_{p_i \in {}^\circ t_u} (\text{ord}(t_u) + M_0(p_i) \times |\sigma|)$$

The quantity q is then deduced from the previous relations and equal to:

$$q = \max_{\forall t_u} (n_{tu})$$

For a repetitive firing sequence σ , the duration of a cycle, denoted by π_{obj} , is given by the following expression:

$$\pi_{obj} = \sum_{i=1}^{|\sigma|} x_i$$

The following linear programs then provide the functioning bounds of the repetitive sequence σ , the network being assumed to be relaxed (the quantities c_{0k} , d_{0k} , $\forall k$ do not create any restriction in this case [BON 01a]):

$$\begin{aligned} \mu_\sigma^{\min} &= \min(\pi) \quad (\mu_\sigma^{\max} = \max(\pi)) \\ \text{with } \sum_{i=1}^{|\sigma|} x_i, \text{ subject to the constraints of system } S_\sigma(q). \end{aligned}$$

In order to obtain a periodic schedule, associated with the firing sequence σ , with a fixed cycle time $\pi_{obj} \in [\mu_\sigma^{\min}, \mu_\sigma^{\max}]$, the following system has to be solved:

$$\sum_{i=1}^{|\sigma|} x_i = \pi_{obj}, \text{ subject to the constraints of the system } S_\sigma(q).$$

Note: the consequence of the cyclic feature of the expressed constraints (due to the fact that the considered functioning mode is 1-periodic) is that the determination of the functioning bounds can be achieved by using the system $S(|\sigma| + 1)$, extended with inequalities corresponding to the tokens for which the consumption occurs later. The system $S_\sigma(q)$ can be rewritten under this following form:

$$S_\sigma(q) : \begin{cases} c_{0k} \leq x_1 \leq d_{0k} \forall k, \quad 1 \leq k \leq n \\ \max(c_{02} - x_1, c_{12}) \leq x_2 \leq \min_{k=2, \dots, n}(d_{0k} - x_1, d_{1k}) \\ \vdots \\ \max\left(\max_{j=0, \dots, q-2}\left(c_{jq} - \sum_{s=j+1}^{q-1} x_s\right), c_{q-1,q}\right) \leq x_q \\ x_q \leq \min\left(\min_{\substack{j=0, \dots, q-2 \\ k=q, \dots, n}}\left(d_{jk} - \sum_{s=j+1}^{q-1} x_s\right), \min_{k=q, \dots, n}(d_{q-1,k})\right) \end{cases}$$

with $\forall k, c_{0k} = 0$ and $d_{0k} = +\infty$ and $\forall(j, k) \in [0, q-1] \times [1, q], j \notin \text{SEN}(q)$ and $k \neq q$, then $c_{jk} = 0$.

The specific structure of the system points out clearly the relations existing between some firing instants. The choice of a value for the q th firing instant (i.e. x_q) could then depend on the chosen values for the $(q-1)$ th previous firing instants.

Application to a graphite ceramic workshop

The investigation of the particular functioning $\sigma = t_1 t_2 t_3 t_4 t_5 t_6$ of length $|\sigma| = 6$ leads to the following system:

$$S_\sigma(7) : \begin{cases} 0 \leq x_1 \leq +\infty \\ 0 \leq x_2 \leq +\infty \\ 4 \leq x_3 \leq 10 \\ 0 \leq x_4 \leq +\infty \\ 0 \leq x_1 + x_2 + x_3 + x_4 \leq +\infty \\ 4 \leq x_5 \leq 6 \\ 8 \leq x_6 \leq 720 \\ 0 \leq x_4 + x_5 + x_6 + x_1 \leq +\infty \\ 60 \leq 5 \times (x_1 + x_2 + x_3 + x_4 + x_5 + x_6) + x_2 \leq 130 \\ 2 \leq x_6 + x_1 + x_2 \leq +\infty \\ 8 \leq x_4 + x_5 + x_6 + x_1 + x_2 + x_3 + x_4 \leq 36 \end{cases}$$

This system is obtained with $M^T = [5, 0, 1, 0, 1, 1, 1, 0, 1]$. It corresponds to the marking of one of the state classes associated with the considered functioning. Performance evaluation can be computed by the following linear programs:

$$\mu_\sigma^{\min} = \min(\pi) \quad (\mu_\sigma^{\max} = \max(\pi))$$

with $\sum_{i=1}^6 x_i$, subject to the constraints of $S_\sigma(7)$.

The performances for the sequence $\sigma = t_1 t_2 t_3 t_4 t_5 t_6$ are:

$$\begin{aligned}\mu_\sigma^{\min} &= 16 : ([x_1, \dots, x_6] = [0, 0, 4, 0, 4, 8]), \\ \mu_\sigma^{\max} &= 26 : ([x_1, \dots, x_6] = [0, 0, 10, 0, 6, 10]).\end{aligned}$$

A periodic schedule for a desired period can then be computed. The next section is devoted to the synthesis of a dynamic robust control when disturbances occur on the planned instants of this periodic reference schedule.

13.4. Step 2: dynamic phase

13.4.1. Temporal flexibility

The occurrence of a disturbance on the i th firing instant x_i (the i th fired transition in σ) will be associated with an advance or a delay on the previously scheduled instant. So, the disturbed firing instant will correspond to x_i^{new} with $x_i^{\text{new}} = x_i + \Delta x_i$ ($\Delta x_i < 0$ representing an advance and $\Delta x_i > 0$ a delay, on the previously scheduled instant x_i). The admissibility of a disturbance x_i on the i th firing instant, for a particular schedule is verified if and only if:

$$\begin{aligned}&\max \left(\max_{j=0, \dots, i-2} \left(c_{ji} - \sum_{s=j+1}^{i-1} x_s \right), c_{i-1,i} \right) - x_i \leq \Delta x_i \\ &\leq \min \left(\min_{\substack{j=0, \dots, i-2 \\ k=i, \dots, n}} \left(d_{jk} - \sum_{s=j+1}^{i-1} x_s \right), \min_{k=i, \dots, n} (d_{i-1,k}) \right) - x_i\end{aligned}$$

Δx_i depends on the previous choices on the transition firing instants.

Thus, the occurrence of an admissible disturbance Δx_i on the i th firing instant x_i will not produce any operation duration violation for the system considered

if an actualization of the following firing instant (the post-disturbance ones) depending on the i th one is feasible. This actualization refers to the firing instants $x_{i+1}, x_{i+2}, \dots, x_{|\sigma|}$, but also x_1, x_2, \dots, x_{i-1} , these last quantities being relative to the period following the one during which the disturbance appeared. Indeed, the constraints considered are cyclic due to the periodic functioning mode. So, if a disturbance occurs on the i th firing instant x_i , the firing instants $x_{i+1}, x_{i+2}, \dots, x_{|\sigma|}, x_1, x_2, \dots, x_{i-1}$, will be referred to as post-disturbance instants. The post-disturbance instants that are coming after the firing instant x_i must verify the following new conditions:

$$\max \left(\max_{j=0, \dots, u-3} \left(c_{ju} - \sum_{s=j+1}^{u-1} x_s - \sum_{r=i}^{u-1} \Delta x_r \right), c_{u-2,u} - (x_{u-1} + \Delta x_{u-1}), c_{u-1,u} \right) \leq x_u^{\text{new}}$$

and

$$x_u^{\text{new}} \leq \min_{\substack{j=0, \dots, u-3, \\ k=u, \dots, n}} \left(d_{jk} - \sum_{s=j+1}^{u-1} x_s - \sum_{r=i}^{u-1} \Delta x_r \right),$$

$$x_u^{\text{new}} \leq \min \left(\min_{k=u, \dots, n} d_{u-2,k} - (x_{u-1} + \Delta x_{u-1}), \min_{k=u, \dots, n} d_{u-1,k} \right)$$

with $u \in \{i+1, \dots, |\sigma|\} \cup \{|\sigma|+1, \dots, |\sigma|+i-1\}$, $i \geq 2$ and $x_{|\sigma|+k}^{\text{new}} = x_k^{\text{new}}$.

To determine a robust schedule that allows smoothing over the maximum disturbance on the i th firing instant, for a given cycle time, the following optimization problem can be defined:

maximize/minimize (Δx_i)

subject to the constraints: $\sum_{r=1}^{|\sigma|} x_r = \pi_{obj},$

$$\max \left(\max_{j=0, \dots, i-2} \left(c_{ji} - \sum_{s=j+1}^{i-1} x_s \right), c_{i-1,i} \right) - x_i \leq \Delta x_i$$

$$\Delta x_i \leq \min \left(\min_{\substack{j=0, \dots, i-2 \\ k=i, \dots, n}} \left(d_{jk} - \sum_{s=j+1}^{i-1} x_s \right), \min_{k=i, \dots, n} (d_{i-1,k}) \right) - x_i$$

$\forall u \in \{i+1, \dots, |\sigma|\} \cup \{|\sigma|+1, \dots, |\sigma|+i-1\}$, $i \geq 2$, with $x_{|\sigma|+r}^{\text{new}} = x_r^{\text{new}}$, $r = 1, i-1$.

$$\max \left(\max_{j=0, \dots, u-3} \left(c_{ju} - \sum_{s=j+1}^{u-1} x_s - \sum_{r=i}^{u-1} \Delta x_r \right), \right.$$

$$\left. c_{u-2,u} - (x_{u-1} + \Delta x_{u-1}), c_{u-1,u} \right) \leq x_u^{\text{new}}$$

and

$$x_u^{\text{new}} \leq \min \left(\min_{\substack{j=0, \dots, u-3 \\ k=u, \dots, n}} \left(d_{jk} - \sum_{s=j+1}^{u-1} x_s - \sum_{r=i}^{u-1} \Delta x_r \right), \right.$$

$$\left. \min_{k=u, \dots, n} d_{u-2,k} - (x_{u-1} + \Delta x_{u-1}), \min_{k=u, \dots, n} d_{u-1,k} \right)$$

and $\forall r \in \{1, \dots, |\sigma|\}$, x_r and x_r^{new} verify $S_\sigma(q)$.

The obtained margins define the firing instant supervision interval, i.e. the admissible temporal domain of the considered event. If the admissibility of the disturbance is verified, the post-disturbance firing instants will be actualized on-line. The control then becomes dynamic, as the approach must be reiterated to determine the margins available on the next firing instant, and so on. In the opposite case, if the transition firing cannot occur in this interval, the disturbance will inevitably lead to a potential constraint violation if the control structure is maintained. These margins give a decisional criterion in order to use post-optimization procedures.

13.4.2. Temporal flexibility and sequential flexibility

The aim is to extend the flexibility level provided by the firing instant approach to obtain flexibility on the sequence order. To increase the margins given by the previous approach, it is possible to evaluate other possible functioning modes by introducing partial order on the considered firing sequence. Then, proactive/reactive control will be treated. The proposed approach, introducing partial order on the sequence, will not only re-actualize the firing instants to compensate the effect of a disturbance, but will also modify the sequence order. Of course, the sequence order to be redefined will concern only the transitions subsequent to the one shifted by the occurrence of the disturbance. Indeed, the previous firing instant approach requires a “strict” ordered sequence. This order is necessary to build the sets TOK and the associated firing inequalities system $S_\sigma(q)$.

Firing dates are expressed as relative values and represent the elapsed time between two firing instants associated with successive transitions in the sequence considered ($x_i \geq 0$ corresponds to the elapsed time between firing of t_{i-1} and t_i). Consequently, a solution of $S_\sigma(q)$ is a set of non-negative instants and performance evaluation is restricted to the sequence considered. The sequence used to establish the associated firing inequalities system $S_\sigma(q)$ will be called the “initial sequence” in what follows. Variable x_i represents the elapsed time between the $(i-1)$ th and the i th firing instant. As the transitions are labeled according to the initial sequence, x_i also represents the elapsed time between t_{i-1} and t_i . Thus, when x_i takes a non-negative value, t_{i-1} is fired before t_i . However, if negative values are permitted, then this order may be permuted: $x_i < 0 \Leftrightarrow t_{i-1}$ is fired after t_i . Introducing partial order on the sequence offers to extend the results of the approach based on firing instants as well in terms of performance evaluation as the dynamic proactive/reactive control.

13.4.2.1. Partial order in performance evaluation

Ensuing from the interpretation of the previous section, as the partial order is obtained by permitting negative firing instants, some new constraints must be added to preserve the semantics of the firing constraints stemming from a definite sequence. This specific sequence, called a reference sequence in what follows, becomes the root of a family of possible schedules.

So, whatever the functioning generated from the reference sequence, the properties to be maintained are:

- The semantics of the cycle time expression $\sum_{i=1}^{|\sigma|=s} x_i = \pi_{obj}$ for the repetitive sequence $\sigma = t_1 t_2 \dots t_i \dots t_s$.

As the firing dates are expressed as relative values and as the cycle time is positive, then it suffices to impose that the last fired transition is transition t_s . Consequently, the reference sequence σ is the root of $(s-1)!$ sequences in the best case.

- The preservation of the same number of firing occurrences of each transition in the reference sequence σ because the generated sequence must be a repetitive one.

Hence, the firing instants x_i with $x_i \in \mathbb{Q}$ must verify the following constraints:

$$\forall j = 1, s-1, \sum_{i=1}^j x_i \leq \sum_{i=1}^s x_i, \quad \forall j = 1, s \sum_{i=1}^j x_i \geq 0$$

As the framework is the repetitive functioning, the set of the firing constraints associated with the reference sequence σ is established by using the marking of one of the state classes belonging to the steady state of the chosen functioning, and moreover for a relaxed system. For each enabled transition t_i , a positivity constraint on its firing instant x_i ($0 \leq x_i \leq +\infty$) is expressed in $S_\sigma(q)$. This constraint represents the possibility for the earliest firing date of each enabled transition to be the instant 0. According to the previously added new constraints, and as quantities c_{jk} and d_{jk} are positive (see section 13.3), it is possible to remove from $S_\sigma(q)$ the positivity constraints on the firing instant(s) of the initially enabled transition(s) without information loss. So, from a reference sequence $\sigma = t_1t_2 \cdots t_i \cdots t_s$ with $|\sigma| = s$ and its associated system $S_\sigma(q)$, the performance evaluation of other possible functioning modes by means of introducing partial order in σ can be stated as follows:

$$\mu_\sigma^{\min} = \min(\pi) \quad (\mu_\sigma^{\max} = \max(\pi)) \quad \text{with } \pi = \sum_{i=1}^{|\sigma|} x_i,$$

subject to:

$$\forall j = 1, |\sigma| - 1, \sum_{i=1}^j x_i \leq \sum_{i=1}^{|\sigma|} x_i, \quad \forall j = 1, |\sigma|, \sum_{i=1}^j x_i \geq 0$$

and

$$S_\sigma(q) - \{0 \leq x_i \leq +\infty, i \mid \forall p \in {}^\circ t_i, M_0(p) \geq \text{Pre}(p, t_i)\}$$

with

$${}^\circ t_i = \{p \in P \mid \text{Pre}(p, t_i) \neq 0\}.$$

Application to a graphite ceramic workshop

For the marking of state class leading to the reference schedule $\sigma = t_1t_2t_3t_4t_5t_6$, transitions t_1, t_2 and t_4 are enabled. Thus, the linear programs considered to compute performance evaluation are:

$$\mu_\sigma^{\min} = \min(\pi) \quad (\mu_\sigma^{\max} = \max(\pi)) \quad \text{with } \pi = \sum_{i=1}^6 x_i$$

subject to:

$$\left\{ \begin{array}{l} 4 \leq x_3 \leq 10 \\ 4 \leq x_5 \leq 6 \\ 8 \leq x_6 \leq 720 \\ 0 \leq x_1 + x_2 + x_3 + x_4 \leq +\infty \\ 4 \leq x_4 + x_5 + x_6 + x_1 \leq +\infty \\ 60 \leq 5 \times (x_1 + x_2 + x_3 + x_4 + x_5 + x_6) + x_2 \leq 130 \\ 2 \leq x_6 + x_1 + x_2 \leq +\infty \\ 8 \leq x_4 + x_5 + x_6 + x_1 + x_2 + x_3 + x_4 \leq 36 \\ \forall j = 1, 5, \quad \sum_{i=1}^j x_i \leq \sum_{i=1}^6 x_i \\ \forall j = 1, 6, \quad \sum_{i=1}^j x_i \geq 0 \end{array} \right.$$

The results are:

$$\begin{aligned} \mu_\sigma^{\min} &= 12 : \sigma^{\min} = t_1 t_2 t_4 t_3 t_5 t_6 : [0, 0, 4, -4, 4, 8] \\ \mu_\sigma^{\max} &= \frac{65}{2} : \sigma^{\max} = t_2 t_3 t_4 t_5 t_1 t_6 : \left[\frac{65}{2}, -\frac{65}{2}, 10, \frac{7}{2}, 6, 13 \right] \end{aligned}$$

Notice that the previous sequences are different from one another and furthermore from the reference sequence $\sigma = t_1 t_2 t_3 t_4 t_5 t_6$. The cycle time interval is larger than that obtained for the reference sequence ($[16, 26] \subset [12, 65/2]$).

13.4.2.2. Partial order in proactive/reactive control

As seen previously (see section 13.4.1), the firing instants actualization due to a disturbance (*delay/advance in regard of the scheduled instant*) must preserve the reference firing sequence leading to time flexibility. Using partial order, it is possible not only to actualize the firing instants to compensate for any disturbance effect, but also to modify the firing sequence (*order flexibility*).

To express the constraints to be considered for the actualization of firing instants, the sequence associated with the current schedule is as follows: t_1 is the first transition of the sequence, t_s the last transition, and the transitions are indexed in regard of the firing sequence: $t_1 t_2 \dots t_i t_{i+1} \dots t_s$.

Let us denote by:

- x_i^{ref} : the planned firing instant of transition t_i for the current schedule.
- x_i : the effective firing instant of transition t_i . $x_i = x_i^{\text{ref}} + \Delta x_i$ with Δx_i an admissible delay (advance) ($\Delta x_i^{\min} \leq \Delta x_i \leq \Delta x_i^{\max}$).

When using order flexibility, the next actualized firing instant x_k^{new} must respect the following constraints along with those expressed in (13.4):

- 1) $\sum_{k=1}^i x_k \leq \sum_{k=1}^j x_k^{\text{new}}, \forall j \in \{i+1, \dots, s\}.$
- 2) $\sum_{k=1}^j x_k^{\text{new}} \leq \min_{n=i+1, s} (\sum_{k=1}^n x_k^{\text{new}}), \forall j \in \{1, \dots, i\}.$

The first constraint lays down that the next actualized firing instants must be greater than the current instant (firing instant of transition t_i) while allowing permutations in the order of the transitions still awaiting firing (i.e $t_{i+1} \dots t_s$). The second constraint allows a new order for the transitions already fired ($t_1 \dots t_i$), but for the next cycle.

Moreover, the next actualized firing instants must respect the constraints of a periodic schedule to ensure a good management of the work-in progress.

$$3) 0 \leq \sum_{k=1}^j x_k^{\text{new}} \leq \sum_{k=1}^s x_k^{\text{new}}, \forall j \in \{1, \dots, s-1\} \text{ with } \sum_{k=1}^s x_k^{\text{new}} = \pi_{\text{new}}$$

The quantities x_k^{new} are obtained by a linear program that maximizes the supervision interval of the next fired transition. As this next transition is unknown *a priori* when several transitions are enabled by the current marking, an equal number of linear programs as enabled transitions must be solved to select each of the transitions in turn by time progression.

13.5. Restrictions due to p-time PNs

The p-time PNs offer a powerful graphic way to naturally represent the constraints of resource availabilities by means of synchronization structures. Nevertheless, this model is not suitable for representing task preemption. Thus, the proposed approach is more appropriate for systems without β_1 .

The robust proactive/reactive approach presented, initially developed to treat the repetitive functioning has been extended to transient phases leading to a repetitive functioning: starting phase of a production system [CAL 04] (and conversely from the periodic functioning mode to unload the system), transient functioning during the maintenance phase [CAL 05].

For the starting phase leading to a repetitive functioning, the model is assumed to be non-relaxed to build the firing inequalities system $S(q)$. Some constraints are added to this system to express that the firing instants of the repetitive reference sequence are relevant if they are greater than the transient mode duration.

For transient functioning during the maintenance phase, according to the previous restriction, p-time PNs remain efficient if preventive maintenance is considered as the temporary loss of some unoccupied resources, regarded as a non-preemptive task and not requiring a stochastic model. To avoid down time of the manufacturing system, the policy generally consists of fitting a maintenance program into the production schedule. This decreases the number of the available resources required to continue the production and causes a transient functioning during the maintenance phase. At step 0, as the possible states of a resource are “free”, “operating” or “in maintenance”, the model is completed to represent the unavailability state due to preventive maintenance. Thus, some places and choice structures must be added. For instance, for the illustrative example, the initial model described in Figure 13.2 becomes the model described in Figure 13.4.

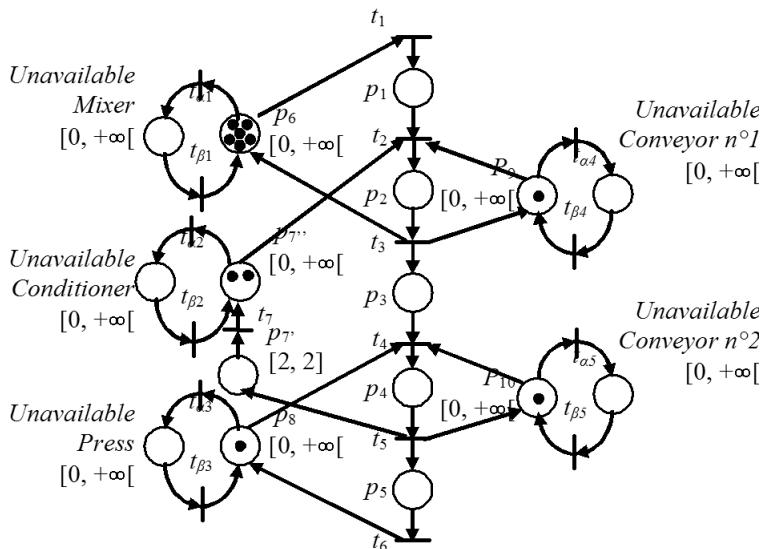


Figure 13.4. Maintenance model

Consequently, fitting a preventive maintenance plan into a busy production schedule will be tantamount to determining the firing instants of the following sequence $\sigma = (\sigma_r \sigma_t \sigma'_r)$, where

σ_r is the reference periodic sequence of the current functioning, σ_t is the sequence associated with the preventive maintenance phase, and σ'_r is the reference periodic sequence of the new functioning.

So, a firing inequalities system is built from this sequence. Constraints are added to ensure the good management of the work-in process, and to express that the firing instants of the preventive maintenance sequence are relevant if they are greater than those of the current functioning, but less than those of the new functioning.

13.6. Bibliography

- [AYG 03] AYGALINC P., CALVEZ S. and BONHOMME P., "Using firing instants approach and partial order to control time critical", *Proceedings of the 9th IEEE ETFA03*, vol. 1, Lisbon, Portugal, p. 82–89, 2003.
- [BER 83] BERTHOMIEU B. and MENASCHE M., "An enumerative approach for analyzing time Petri nets", *IFIP Congress Series*, vol. 9, Elsevier Science Publ. Comp. (North Holland), p. 41–46, 1983.
- [BON 01a] BONHOMME P., AYGALINC P. and CALVEZ S., "Control and performances evaluation using an enumerative approach", *5th World Multiconference on Systemics, Cybernetics and Informatics, SCI 2001*, Orlando, Florida, USA, July 2001.
- [BON 01b] BONHOMME P., AYGALINC P. and CALVEZ S., "Robust control for time critical systems", *8th IEEE ETFA 01*, Antibes, France, 2001.
- [BOU 93] BOUCHENEH B. and BERTHELOT G., "Towards a simplified building of time Petri nets reachability graph", *Proceedings of PNPM'93*, p. 46–55, September 1993.
- [BRA 82] BRAMS G.W., *Reseaux de Petri: Théorie et Pratique. Tome 1: Théorie et Analyse; Tome 2: Modélisation et Applications*, Masson, Paris, 1982.
- [CAL 97] CALVEZ S., AYGALINC P. and KHANSA W., "P-time Petri Nets for manufacturing systems with staying time constraints", *CIS 97*, Belfort, p. 495–500, May 1997.
- [CAL 04] CALVEZ S., AYGALINC P. and BONHOMME P., "Proactive/reactive approach for transient functioning mode of time critical systems", *MCPL 2004*, Santiago, Chile, p. 65–70, November 2004.
- [CAL 05] CALVEZ S., AYGALINC P. and BONHOMME P., "Proactive/reactive approach for maintenance tasks in time critical systems", *Proceedings of the 10th IEEE ETFA 05*, Catania, Italy, September 2005.
- [DAV 92] DAVID R. and ALLA H., *Du Grafcet aux Réseaux de Petri*, Hermès, Paris, 1992.
- [DIA 94] DIAZ M. and SENAC P., "Time stream Petri Nets: a model for timed multimedia information", *Proceedings of the 15th ICATPN*, vol. 815 of *Lecture Notes in Computer Sciences*, Springer-Verlag, Saragossa, Spain, p. 219–238, June 1994.

- [KHA 96] KHANSA W., AYGALINC P. and DENAT J.-P., “Structural analysis of p-time Petri nets”, *CESA'96 IMACS Multiconference*, Lille, p. 127–136, July 1996.
- [KHA 97] KHANSA W., Réseaux de petri P-temporels: contribution à l'étude des systèmes à événements discrets, PhD Thesis, University of Savoy, Annecy, 1997.
- [MER 74] MERLIN P., A study of the recoverability of communication protocols, PhD Thesis, Computer Science Dep., University of California, Irvine, 1974.
- [MUR 89] MURATA T., “Petri nets: properties, analysis and applications”, *Proc. of the IEEE*, vol. 77, no. 4, p. 541–579, 1989.
- [RAM 74] RAMCHANDANI C., Analysis of asynchronous concurrent systems by timed Petri Nets, PhD Thesis, MIT, Dept. Electrical Engineering, Cambridge, Mass., 1974.
- [SIF 77] SIFAKIS J., “Use of Petri nets for performance evaluation”, in BEILNER H. and GELENBE E. (Eds.), *Modelling and Performance Evaluation of Computer Systems, Measuring, Modelling and Evaluating Computer Systems*, Amsterdam: North Holland, p. 75–93, 1977.
- [VAN 92] VAN DER AALST W.M.P., Timed coloured Petri nets and their application to logistics, PhD Thesis, Eindhoven University of Technology, Eindhoven, 1992.

Chapter 14

Small Perturbations on Some NP-Complete Scheduling Problems

We consider some basic NP-complete scheduling problems in which we add both a feasible solution and a small perturbation to the instance (typically, removing one of the precedence constraints, decreasing or increasing by one unit of a numerical parameter, etc.). The aim is to compute a feasible schedule for the perturbed problem, using the schedule of the unperturbed problem which is given with the problem instance. It is proved that this perturbed problem is NP-complete for some basic scheduling problems (single machine scheduling, multiprocessor scheduling, scheduling with communication delays, etc.). Thus, it is shown that for these problems the knowledge of a solution cannot be efficiently used to obtain a schedule for the perturbed problem.

14.1. Introduction

The goal of this chapter is to study, for some basic NP-complete scheduling problems, the way a feasible solution can be used to compute a schedule when a *small* perturbation occurs on the data. The perturbations considered here are *minimal*: for example, they consist of removing a unique precedence constraint, or removing a single task, or in the unitary increasing, or decreasing, of a sole numerical parameter. Furthermore, the perturbations taken into account are discrete (the variations on the

Chapter written by Christophe PICOULEAU.

numerical parameters are integer). Scheduling problems where the perturbations of the numerical parameters are continuous are taken into account in [HAL 04]. The reader can refer to section 4.2.1, which gives the main results of this study. The perturbations considered are not obvious in the sense that the answer to the decision problems is not always “yes”. It is proved that despite the minimality of the perturbations it is not always easy to efficiently use the schedule provided before the perturbation. Indeed it has been proved that the NP-completeness of this kind of *perturbation problems* issued from some well-known NP-complete problems of scheduling theory (see [GAR 79, PIC 95]): sequencing with release times and deadlines; multiprocessor scheduling for any fixed number of processors $m, m \geq 2$; unitary execution times (UET) schedule of a precedence graph on m processors; unitary execution times and unit communication times (UET-UCT) schedule of a precedence graph on an unbounded number of processors.

14.2. Problem definitions

We give here the definitions of the NP-complete scheduling problems that will be considered in this chapter. We also define formally the perturbed versions of these problems: an instance of a perturbation problem is obtained by adding both a feasible schedule and the perturbation of one parameter to the former instance.

14.2.1. Sequencing with release times and deadlines

The sequencing with release times and deadlines decision problem, henceforth called SRTD, is defined as follows:

INSTANCE: a set T of n tasks, and for each task $i \in T$: a processing time $p_i \in \mathbb{N}$, a release time $r_i \in \mathbb{N}$, and a deadline $d_i \in \mathbb{N}$.

QUESTION: is there a single processor non-preemptive schedule for T that satisfies the release times and deadline constraints? This problem is NP-complete in the strong sense (see [GAR 79]).

The SRTD with a unitarily increasing single release time, denoted by SRTDr, is defined as follows:

INSTANCE: a set T of n tasks, for each task $i \in T$: a processing time $p_i \in \mathbb{N}$, a release time $r_i \in \mathbb{N}$, and a deadline $d_i \in \mathbb{N}$; S a schedule satisfying the release times and the deadlines; a task $t \in T$.

QUESTION: is there a single processor non-preemptive schedule that satisfies the release times and deadlines where the new release time for the task t is $r_t + 1$ (the other parameters stay unchanged)?

Symmetrically, the SRTD with a unitarily decreasing deadline, denoted by SRTDd, is defined as follows:

INSTANCE: a set T of n tasks, for each task $i \in T$, a processing time $p_i \in \mathbb{N}$, a release time $r_i \in \mathbb{N}$, and a deadline $d_i \in \mathbb{N}$; S a schedule satisfying the release times and the deadlines; a task $t \in T$.

QUESTION: is there a single processor, non-preemptive schedule that satisfies the release times and the deadline constraints in which the new deadline for the task t is $d_t - 1$?

Note: the two problems above are equivalent since by reversing the time axis, SRTDr can simply be transformed into SRTDd and vice versa.

The SRTD with a unitarily increasing processing time, denoted by SRTDp, is defined as follows:

INSTANCE: a set T of n tasks, for each task $i \in T$, a processing time $p_i \in \mathbb{N}$, a release time $r_i \in \mathbb{N}$, and a deadline $d_i \in \mathbb{N}$; S a schedule satisfying the release times and the deadlines; a task $t \in T$.

QUESTION: is there a single processor, namely preemptive schedule that satisfies the release times and the deadlines when the new processing time for the task t is $p_t + 1$?

14.2.2. Multiprocessor scheduling

The multiprocessor scheduling problem, called MS, is defined as follows:

INSTANCE: a set T of n tasks, for each task $i \in T$, a processing time $p_i \in \mathbb{N}$; m the number of identical processors; $D \in \mathbb{N}$ the overall deadline.

QUESTION: is there a m -processor non-preemptive schedule for T that meets the overall deadline D ?

This problem is NP-complete for every fixed $m \geq 2$ and NP-complete in the strong sense when m is a parameter (see [GAR 79]).

The multiprocessor scheduling with a unitary decreasing of one of the processing times for $m = 2$ processors, denoted by MSp^{-2} is defined as follows:

INSTANCE: a set T of n tasks, for each task $i \in T$, a processing time $p_i \in \mathbb{N}$; an overall deadline D ; S a two processor schedule of T with length less than or equal to D ; a task $t \in T$.

QUESTION: is there a two processor schedule of length $D - 1$ when the processing time of t becomes $p_t - 1$ and all other processing times remain unchanged?

The multiprocessor scheduling with a unitarily increasing processing time for $m = 2$ processors, denoted by MSp^{+2} is defined as follows:

INSTANCE: a set T of n tasks, for each task $i \in T$, a processing time $p_i \in \mathbb{N}$; an overall deadline D ; S a two processor schedule of T with length less than or equal to D ; a task $t \in T$.

QUESTION: is there a two processor schedule of length D when the processing time of t becomes $p_t + 1$ and all other processing times remain unchanged?

14.2.3. Unit execution times scheduling

The unit execution times scheduling decision problem on m processors, named UET is defined as follows:

INSTANCE: $G = (I, A)$ a directed acyclic graph; m and B two positive integers.

QUESTION: is there a schedule of G on m processors with length less than or equal to B ?

For this problem, a schedule is a mapping S of the set of tasks I to $\{1, \dots, B\} \times \{1, \dots, m\}$ such that: for two tasks i and j , $i \neq j$, $S(i) = (t_i, \pi_i) \neq S(j) = (t_j, \pi_j)$ (at each time period each processor executes at most one task) and m tasks at most can be processed during an equivalent time period; for each arc (i, j) of A , $t_i < t_j$ (an arc (i, j) is also a precedence constraint, so i precedes j).

It is well known that this problem is NP-complete in the strong sense even for $B = 3$ (see [GAR 79, LEN 78]).

The $\text{UET}^{-(i,j)}$ problem (one precedence constraint is deleted) is defined as follows:

INSTANCE: $G = (I, A)$ a directed acyclic graph, $(i, j) \in A$ an arc of G ; m, B two positive integers; S a UET schedule of G on m processors with length less than or equal to B .

QUESTION: is there a UET schedule of $G^{-(i,j)} = (I, A \setminus \{(i, j)\})$ on m processors with length less than or equal to $B - 1$?

The UET $^{-i}$ problem (one task is deleted) is defined as follows:

INSTANCE: $G = (I, A)$ a directed acyclic graph, $i \in I$ a task of G ; m and B two positive integers; S a UET schedule of G on m processors with length less than or equal to B .

QUESTION: is there a UET schedule of $G^{-i} = (I \setminus \{i\}, A')$ on m processors with length less than or equal to $B - 1$? (A' is the subset of arcs a of A such that i is not an endpoint of a .)

14.2.4. Scheduling unit execution times with unit communication times

The UET-UCT scheduling problem is defined as follows:

INSTANCE: $G = (I, A)$ a directed acyclic graph and B a positive integer.

QUESTION: is there a schedule with communication of G with length less than or equal to B ?

A schedule with communication is a mapping S of the set of tasks I to $\{1, \dots, B\} \times \mathbb{N}^*$ such that: for two distinct tasks i and j , $S(i) = (t_i, \pi_i) \neq S(j) = (t_j, \pi_j)$ (for each time period a processor executes at most one task); for each arc (i, j) of A , if $\pi_i = \pi_j$ then $t_i < t_j$ and if $\pi_i \neq \pi_j$ then $t_i + 1 < t_j$ (if i and j are performed by the same processor then there is no communication between the two tasks: so $t_i < t_j$, otherwise the task j needs a unitary communication from i and $t_i + 1 < t_j$).

This problem has been proved NP-complete in the strong sense in [PIC 95].

The UET-UCT $^{-(i,j)}$ problem (one precedence constraint is deleted) is defined as follows:

INSTANCE: $G = (I, A)$ a directed acyclic graph, $(i, j) \in A$ an arc of G ; B a positive integer; S a UET-UCT schedule of G with length less than or equal to B .

QUESTION: is there a UET-UCT schedule of $G^{-(i,j)} = (I, A \setminus \{(i,j)\})$ with length less than or equal to $B - 1$?

The UET-UCT $^{-i}$ problem (one task is deleted) is defined as follows:

INSTANCE: $G = (I, A)$ a directed acyclic graph, $i \in I$ a task of G ; B a positive integer; S an UET-UCT schedule of G with length less than or equal to B .

QUESTION: is there a UET-UCT schedule of $G^{-i} = (I \setminus \{i\}, A')$ with length less than or equal to $B - 1$?

14.3. NP-completeness results

Before proving the results for the problems SRTDr and SRTDd, it will be shown the NP-completeness of a problem issued from SUBSET-SUM (see [GAR 79] for the definition of SUBSET-SUM). The problem named SUBSET-SUM-1 is defined as follows:

INSTANCE: m items of size $s(b_i) \in \mathbb{N}$, $1 \leq i \leq m$; $B \in \mathbb{N}$; and $I \subset \{1, \dots, m\}$ such that $\sum_{i \in I} s(b_i) = B$.

QUESTION: is there $K \subset \{1, \dots, m\}$ such that $\sum_{i \in K} s(b_i) = B - 1$?

We recall the definition of the NP-complete problem PARTITION (see [GAR 79]):

INSTANCE: k items a_1, \dots, a_k of size $s(a_j) \in \mathbb{N}^*$, $1 \leq j \leq k$.

QUESTION: is there $J \subset \{1, \dots, k\}$ such that $\sum_{j \in J} s(a_j) = \frac{1}{2} \sum_{j=1}^k s(a_j)$ (w.l.o.g. it is supposed that $\sum_{j=1}^k s(a_j)$ is even)?

LEMMA 14.1.– SUBSET-SUM-1 is NP-complete.

Proof. SUBSET-SUM-1 \in NP. The transformation is from PARTITION. Set $m = k + 1$; $s(b_i) = s(a_i)$, $1 \leq i \leq m - 1$ and $s(b_m) = 1 + \frac{1}{2} \sum_{j=1}^k s(a_j) = B$; $I = \{m\}$.

If there is $J \subset \{1, \dots, k\}$ such that $\sum_{j \in J} s(a_j) = \frac{1}{2} \sum_{j=1}^k s(a_j)$ then setting $J = K$ we have $\sum_{i \in K} s(b_i) = B - 1$. Conversely, if there is $K \subset \{1, \dots, m\}$ such that $\sum_{i \in K} s(b_i) = B - 1$, $m \notin K$, so setting $J = K$ we obtain $\sum_{j \in J} s(a_j) = \frac{1}{2} \sum_{j=1}^k s(a_j)$. \square

THEOREM 14.1.– SRTDr is NP-complete.

Proof. SRTDr \in NP. The transformation is from SUBSET-SUM-1. We denote $W = \sum_{i=1}^m s(b_i)$. We set $n = m + 1$, the tasks have processing times $p_i = s(b_i)$, $1 \leq i \leq n - 1$, $p_n = 1$; the release times are $r_i = 0$, $i \leq n - 1$, $r_n = W - B$; the deadlines are set to $d_i = W + 1$, $i \leq n - 1$, $d_n = W - B + 2$; the task t is $t = n$; the feasible schedule S is obtained by successively scheduling the tasks $i \notin I$ from date 0, the task n with the starting time $W - B$ and the tasks $i \in I$ from time $W - B + 1$ to $W + 1$.

If there is $K \subset \{1, \dots, m\}$ such that $\sum_{i \in K} s(b_i) = B - 1$, a feasible schedule is obtained by successively scheduling the tasks $i \notin K$ during the time interval $[0, W - B + 1]$, the task n with the starting time $W - B + 1$ and the tasks $i \in K$ from time $W - B + 2$ to $W + 1$. Conversely, if there is a feasible schedule for T , the task n is scheduled with starting time $W - B + 1$ since his new release time is $W - B + 1$. Thus, the sum of the processing times of the tasks scheduled after n is $B - 1$, so K exists such that $\sum_{i \in K} s(b_i) = B - 1$. \square

COROLLARY 14.1.– SRTDd is NP-complete.

Proof. Reversing the time axis, SRTDr and SRTDd are equivalent. \square

COROLLARY 14.2.– SRTDp is NP-complete.

Proof. SRTDp \in NP. The transformation is from SUBSET-SUM-1. We denote $W = \sum_{i=1}^m s(b_i)$. We set $n = m + 1$, the processing times of the tasks are: $p_i = s(b_i)$, $1 \leq i \leq n - 1$, $p_n = 1$; the release times are $r_i = 0$, $i \leq n - 1$, $r_n = W - B + 1$; the deadlines are set to $d_i = W + 2$, $i \leq n - 1$, $d_n = W - B + 3$; the task t is $t = n$; the feasible schedule S is obtained by successively scheduling the tasks $i \notin I$ from the date 0, the task n with the starting time $W - B + 1$ and the tasks $i \in I$ from time $W - B + 2$ to $W + 2$.

If there is $K \subset \{1, \dots, m\}$ such that $\sum_{i \in K} s(b_i) = B - 1$, a feasible schedule is obtained by successively scheduling the tasks $i \notin K$ during the time interval $[0, W - B + 1]$, the task n with the starting time $W - B + 1$ and the tasks $i \in K$ from time $W - B + 3$ to $W + 2$. Conversely, if there is a feasible schedule for T , the task n is scheduled during the time interval $[W - B + 1, W - B + 3]$ since its new processing

time is $p_t = 2$. Thus, the sum of the processing times of the tasks scheduled after n is $B - 1$, so K exists such that $\sum_{i \in K} s(b_i) = B - 1$. \square

COROLLARY 14.3. – MSp^-2 is NP-complete.

Proof. $\text{MSp}^-2 \in \text{NP}$. We use a polynomial transformation from PARTITION defined above. W.l.o.g. it is supposed that $\sum_{j=1}^k s(a_j)$ is even and $s(a_j) > 1, 1 \leq j \leq k$.

An instance of MSp^-2 is built as follows: the number of tasks is set to $n = k + 2$; the processing times are $p_i = s(a_i)$ for $1 \leq i \leq k$, $p_{k+1} = B$, and $p_{k+2} = B - 1$ where $B = \frac{1}{2} \sum_{j=1}^k s(a_j)$; $D = 2B$; $t = k + 1$; thus, a schedule S of length D is obtained by processing the tasks $k + 1$ and $k + 2$ on the same processor and the k other tasks on the second processor.

If the processing time of t becomes $p_{k+1} - 1 = B - 1$ and the new deadline is $D - 1 = 2B - 1$, necessarily the tasks $k + 1$ and $k + 2$ must be scheduled by distinct processors (recall that $s(a_j) > 1, 1 \leq j \leq k$). So, it is easy to verify that the answer to such an instance of MSp^-2 is equivalent to answering the corresponding instance of PARTITION. \square

COROLLARY 14.4. – MSp^+2 is NP-complete.

Proof. The proof is similar to the proof above. $\text{MSp}^+2 \in \text{NP}$. We use a transformation from PARTITION.

An instance of MSp^+2 is built as follows: the number of tasks is $n = k + 2$; the processing times are $p_i = s(a_i)$ for $1 \leq i \leq k$, $p_{k+1} = B$, and $p_{k+2} = B + 1$ where $B = \frac{1}{2} \sum_{j=1}^k s(a_j)$; $D = 2B + 1$; $t = k + 1$; a schedule S of length D is obtained by processing the tasks $k + 1$ and $k + 2$ on the same processor and the k other tasks on the second processor.

If the processing time of t becomes $p_{k+1} + 1 = B + 1$ and the deadline keeps the value $D = 2B + 1$, the tasks $k + 1$ and $k + 2$ are scheduled by distinct processors. So the answer to such an instance of MSp^+2 is equivalent to answering the corresponding instance of PARTITION. \square

COROLLARY 14.5. – $\text{UET}^{-(i,j)}$ is NP-complete.

Proof. The $\text{UET}^{-(i,j)}$ problem belongs to NP. The polynomial transformation (inspired from [LEN 78]) is from the NP-complete problem CLIQUE defined as follows:

INSTANCE: $H = (V, E)$ a connected undirected graph, $k \leq |V|$ a positive integer.

QUESTION: does H contain a complete subgraph with k vertices?

We will use the following notations: $\theta = \frac{k(k-1)}{2}$, $\nu = |E| - \theta$ (w.l.o.g. it is supposed that $|E| > \theta$) and $\mu = \max\{k, |V| - k + \theta, \nu\}$.

The task graph G is built as follows (see Figure 14.1): each vertex x of H is associated with a vertex-task x ; each edge $\{x, y\}$ is associated with an edge-task $\{x, y\}$; each edge-task $\{x, y\}$ has for predecessors the two vertex-tasks x and y ; each edge-task has to precede a terminal task t ; to complete the description of G we add a path (p_1, p_2, p_3, p_4) with p_4 preceding the task t , and we also add $\mu - k$ tasks $t_1^1, \dots, t_{\mu-k}^1$ preceding the task p_2 , $\mu - |V| + k - \theta$ tasks $t_1^2, \dots, t_{\mu-|V|+k-\theta}^2$ preceding the task p_3 with the task p_1 as predecessor, and $\mu - \nu$ tasks $t_1^3, \dots, t_{\mu-\nu}^3$ preceding the task p_4 with the task p_2 as predecessor.

We set $m = \mu + 1$, $B = 5$ and $(i, j) = (p_4, t)$.

A schedule of length $B = 5$ for G is made as follows (see Figure 14.1): k vertex-tasks are processed during the time-slot 1, the remaining $|V| - k$ vertex-tasks are processed in the time-slot 2, ν edge-tasks are scheduled in the time-slot 3, the remaining θ edge-tasks are processed in the time-slot 4; the tasks $t_1^1, \dots, t_{\mu-k}^1$ and p_1 are scheduled in the time-slot 1, the tasks $t_1^2, \dots, t_{\mu-|V|+k-\theta}^2$ and p_2 are scheduled in the time-slot 2, the tasks $t_1^3, \dots, t_{\mu-\nu}^3$ and p_3 are processed in the time-slot 3, the task p_4 is processed in the time-slot 4; finally, the terminal task t is processed in the time-slot 5.

We will prove that H contains a clique of size k if and only if $G^{-(p_4, t)}$ can be scheduled on m processors with a makespan less than or equal to $B - 1 = 4$.

First, suppose that H contains a clique of size k . The task graph $G^{-(p_4, t)}$ is scheduled as follows: the tasks $t_1^1, \dots, t_{\mu-k}^1$ and p_1 are scheduled in the time-slot 1, the tasks $t_1^2, \dots, t_{\mu-|V|+k-\theta}^2$ and p_2 are scheduled in the time-slot 2, the tasks $t_1^3, \dots, t_{\mu-\nu}^3$ and p_3 are processed in the time-slot 3, the task p_4 is processed in the time-slot 4; the k vertex-tasks constituting the clique are scheduled in the time-slot 1;

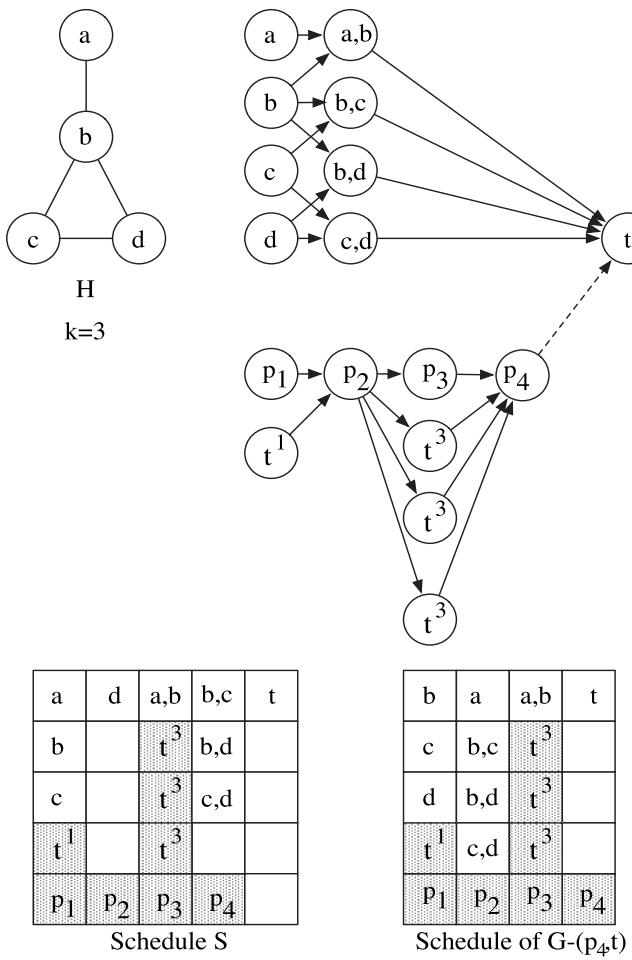


Figure 14.1. The transformation from CLIQUE

the $|V|-k$ other vertex-tasks and the θ edge-tasks occurring in the clique are processed in the time-slot 2; the ν remaining edge-tasks are scheduled in the time-slot 3; the task t is processed in the time-slot 4; thus, we obtain a schedule of length 4.

Now suppose that $G^{-(p_4,t)}$ can be scheduled on m processors by time less than or equal to 4: since p_1, p_2, p_3, p_4 form a chain of length 4, the tasks $t_1^1, \dots, t_{\mu-k}^1$ and p_1 are scheduled in the time-slot 1, the tasks $t_1^2, \dots, t_{\mu-|V|+k-\theta}^2$ and p_2 are scheduled in the time-slot 2, the tasks $t_1^3, \dots, t_{\mu-\nu}^3$ and p_3 are processed in the time-slot 3, and the task p_4 is processed in the time-slot 4; since

$k + \mu - k + 1 + |V| - k + \theta + \mu - |V| + k - \theta + 1 + \nu + \mu - \nu + 1 = 3m$, the terminal task t is necessarily scheduled in the time-slot 4; since each vertex-task has to precede some edge-tasks, and since each edge-task is preceded by two vertex-tasks, k vertex-tasks are scheduled in the time-slot 1 and $\theta = \frac{k(k-1)}{2}$ edge-tasks are scheduled in the time-slot 2: so the corresponding k vertices and θ edges of H form a clique of size k . \square

THEOREM 14.2.— UET-UCT^{-(i,j)} is NP-complete.

Proof. The problem belongs to NP. The polynomial transformation is from the NP-complete problem 3-SAT. We denote by $x_i, 1 \leq i \leq n$, the n Boolean variables and by $C_i, 1 \leq i \leq m$, the set of clauses occurring in 3-SAT.

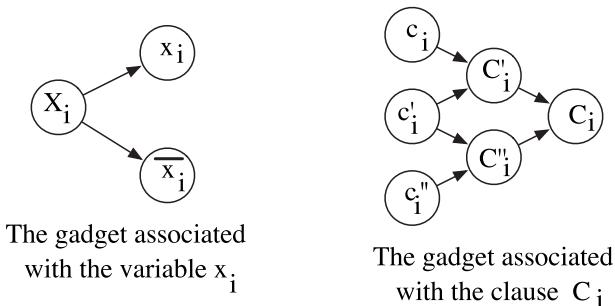


Figure 14.2. The gadgets associated with the variables and with the clauses

The precedence graph G is constructed as follows: each variable x_i is associated with three tasks X_i, x_i, \bar{x}_i ; the task X_i has to precede the two tasks x_i and \bar{x}_i (see Figure 14.2). Each clause $C_i = (a, b, c)$, where a, b, c are three literals, is associated with six tasks $C_i, C'_i, C''_i, c_i, c'_i$ and c''_i ; c_i precedes C'_i , c'_i precedes the two tasks C'_i and C''_i , and the two tasks C'_i and C''_i precede C_i (see Figure 14.2).

Each task c_i, c'_i, c''_i has for predecessor the task associated with the corresponding literal (see Figure 14.3). To complete the description of the precedence graph G , we make two chains (p_1, \dots, p_9) and (q_1, \dots, q_9) , such that their last two tasks p_9, q_9 precede a terminal task t ; the task t is also preceded by the m clause tasks C_i (see Figure 14.4).

Let (i, j) be the arc (q_9, t) , $B = 11$ and S be the UET-UCT schedule of G obtained as follows: X_i, x_i, \bar{x}_i are scheduled on the same processor π_i in the time-slots

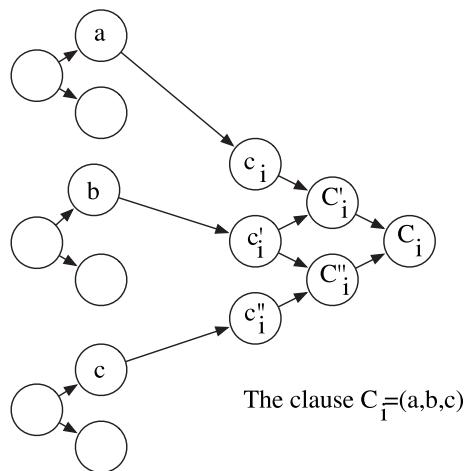


Figure 14.3. The precedence constraints between the variable gadgets and the clause gadgets

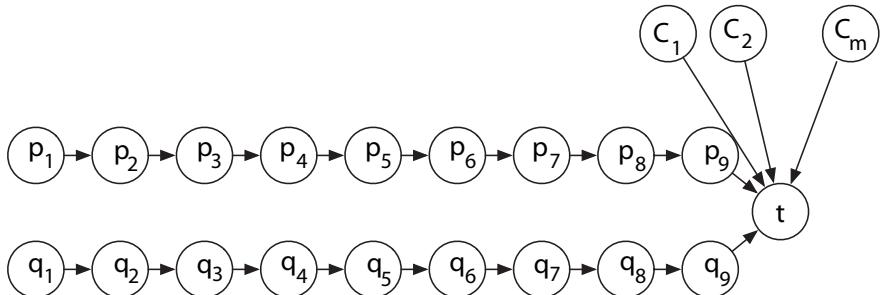


Figure 14.4. The two chains completing G

1, 2, 3, respectively; the six tasks forming the gadget associated with the clause C_i are scheduled on their own set of three processors in the following way: c_i, c'_i, c''_i are processed in the time-slot 5, C'_i, C''_i are processed during the time-slot 7, and the task C_i is scheduled in the time-slot 9; the two chains are scheduled on two processors during the time-slots 1, …, 9, and the terminal task t is performed in the time-slot 11. Thus, we built in polynomial time a feasible UET-UCT schedule of length $B = 11$ for G .

Now, it will be proved that there is a graph schedule $G^{-(i,j)} = (I, A \setminus \{(q_9, t)\})$ with length less than or equal to $B - 1 = 10$ if and only if the corresponding instance of 3-SAT is satisfied.

First, suppose that the instance of 3-SAT is satisfied. We build a schedule in the following way: if the variable x_i has the value *true*, X_i, x_i, \bar{x}_i are scheduled on the same processor π_i in the time-slots 1, 2, 3 respectively; if the variable x_i has the value *false*, X_i, \bar{x}_i, x_i are scheduled on the same processor π_i in the time-slots 1, 2, 3 respectively. Because each clause C_i contains a literal with the value *true*, at least one task among c_i, c'_i, c''_i is scheduled during the time-slot 4, and the other tasks (associated with a literal having the value *false*) are scheduled in the time-slot 5. First suppose that c_i is processed in the time-slot 4 (the case for c''_i is the same): C'_i, C''_i, C_i are processed by the same processor as c'_i in the time-slots 6, 7, 8, respectively. Secondly, suppose that c'_i is processed in the time-slot 4: C'_i is processed by the same processor as c_i in the time-slot 6, C''_i is processed by the same processor as c''_i in the time-slot 6, and C_i is scheduled in the time-slot 8. Since the instance of 3-SAT is satisfied, the tasks C_i are scheduled during the time period 8, the tasks p_1, \dots, p_9, t are scheduled by the same processor in the time-slots 1, ..., 9, 10, respectively, and the tasks q_1, \dots, q_9 are scheduled by the same processor in the time-slots 1, ..., 9. Thus, we obtain a UET-UCT schedule for $G^{-(i,j)} = (I, A \setminus \{(q_9, t)\})$ with length $10 \leq B - 1$.

Now suppose that there is a UET-UCT schedule for $G^{-(i,j)} = (I, A \setminus \{(q_9, t)\})$ with length less than or equal to $B - 1 = 10$. Since t is performed during a time-slot less than or equal to 10, the tasks C_i are completed at time less than or equal to 8. So at least one of the two predecessors of each task C_i is scheduled in a time slot less than or equal to 6. W.l.o.g. it can be supposed that C'_i is scheduled in a time-slot less than or equal to 6, thus at least one task among c_i and c'_i is scheduled in a time-slot less than or equal to 4. W.l.o.g. we suppose that the task c_i is scheduled in a time-slot less than or equal to 4. So its predecessor x_i or \bar{x}_i is processed in the time-slot 2 on the same processor as the corresponding task X_i (X_i is scheduled in the first period). Thus, taking the value *true* for the variables x_i such that the associated tasks x_i are scheduled in the time-slot 2 with the value *false* for the variables x_i , we obtain a truth function that satisfies each clause. \square

THEOREM 14.3.– UET-UCT⁻ⁱ is NP-complete.

Proof. The proof is the same as the proof of theorem 14.2 taking $i = q_9$. \square

14.4. Conclusion

We defined and proved the NP-completeness of some new kinds of scheduling problems. The main feature of these problems is that a feasible schedule is given with a *slight modification* of the data such as the deletion of an arc of the precedence graph, a unitary increasing of one of the release times, etc. The problem is to determine whether the perturbed problem can be easily solved using the solution of the unperturbed problem. It has been shown that even with the knowledge of a solution of the unperturbed problem, obtaining a solution to the perturbed problem is NP-complete.

Numerous other scheduling problems submitted to small perturbations can be studied. For example, we can consider a shop problem (job-shop, flow-shop, open-shop) in which the processing time of a single job is increased or decreased by a single time unit.

For all the non-trivial problems (the answer to the identity problems is not invariably “yes”) studied here, the perturbed version is NP-complete whenever the initial problem is NP-complete. Thus, it will be asked if there exists a NP-complete scheduling problem for which the (non-trivial) perturbed version is polynomial.

In another way, from a practical point of view, a *good solution* is often enough. Thus, in this context we can naturally extend our study to consider how we may efficiently compute a *good* schedule from a *good* schedule when the data of the initial schedule is *slightly* modified? Our intuition is that despite the NP-completeness results proved here, finding a *good* solution could be an easy task from a computational point of view. This study should be interesting for both practical and theoretical reasons.

14.5. Bibliography

- [GAR 79] GAREY M.R. and JOHNSON D.S., *Computers and Intractability, A Guide to the Theory of NP-Completeness*, Freeman, 1979.
- [HAL 04] HALL N.G. and POSNER M.E., “Sensitivity analysis for scheduling problems”, *Journal of Scheduling*, vol. 7, p. 49–83, 2004.
- [LEN 78] LENSTRA J.K. and KAN A.H.G.R., “Complexity of scheduling under precedence constraints”, *Operations Research*, vol. 26, p. 22–35, 1978.
- [PIC 95] PICOULEAU C., “New complexity results on scheduling with small communication delays”, *Discrete Applied Mathematics*, vol. 60, p. 331–342, 1995.

List of Authors

Mohamed Ali ALOULOU
LAMSADE
Paris-Dauphine University
France

Christian ARTIGUES
LAAS-CNRS
University of Toulouse
France

Pascal AYGALINC
LISTIC-Polytech'Savoie
University of Savoy
France

Jean-Charles BILLAUT
Laboratoire d'Informatique-Polytech'Tours
University of Tours
France

Patrice BONHOMME
Laboratoire d'Informatique
University of Tours
France

Cyril BRIAND
LAAS-CNRS
University of Toulouse
France

Soizick CALVEZ
LISTIC-Polytech'Savoie
University of Savoy
France

Philippe CASTAGLIOLA
IRCCyN
University of Nantes
France

Stéphane DAUZÈRE-PÉRÈS
Ecole des Mines de Saint-Etienne
CMP Georges Charpak
Gardanne
France

Marie-Laure ESPINOUSE
G-SCOP
IUT de Grenoble
France

Carl ESSWEIN
Laboratoire d'Informatique-Polytech'Tours
University of Tours
France

Michel GOURGAND
LIMOS
Clermont-Ferrand
France

Nathalie GRANGEON
LIMOS
Clermont-Ferrand
France

Willy HERROELEN
Res. Ctr. Oper. Mgmt.
KU Leuven
Belgium

Marie-José HUGUET
LAAS-CNRS
University of Toulouse
France

Mireille JACOMINO
LAG
ENSIEG-INPG
Grenoble
France

Hoang Trung LA
LAAS-CNRS
University of Toulouse
France

Chams LAHLOU
IRCCyN
Ecole des Mines de Nantes
France

Yann LE QUÉRÉ
LAMIH
University of Valenciennes
France

Roel LEUS
Res. Ctr. Oper. Mgmt.
KU Leuven
Belgium

Pierre LOPEZ
LAAS-CNRS
University of Toulouse
France

Amine MAHJOUB
UTIC
Ecole Supérieure des Sciences et Techniques de Tunis
Tunisia

Aziz MOUKRIM
HeuDiaSyC
University of Technology of Compiègne
France

Sylvie NORRE
LIMOS
Clermont-Ferrand
France

Jonathan E. PECERO SÁNCHEZ
LIG
ENSIMAG Grenoble
France

Christophe PICOULEAU
CEDRIC
CNAM Paris
France

Marie-Claude PORTMANN
LORIA
Ecole des Mines de Nancy
France

Christophe RAPINE
G-SCOP
INP Grenoble
France

André ROSSI
LESTER
South Brittany University
Lorient
France

Bernard ROY
LAMSADE
Paris-Dauphine University
France

Eric SANLAVILLE
LIMOS
Clermont-Ferrand
France

Marc SEVAUX
Lab-STICC
South Brittany University
Lorient
France

Kenneth SÖRENSEN
CIB
KU Leuven
Belgium

Denis TRYSTRAM
LIG
University of Grenoble
France

Index

Symbols

3-SAT, 337, 339

A

approximation, 110, 181, 289
assignment, 25, 89, 90, 220, 236, 271
availability, 21, 70, 80, 265

B

branch-and-bound, 101, 176, 178, 180, 207
branching scheme, 180

C

CLIQUE, 336
coding, 258
communication delay, 273, 281
complexity
 algorithm complexity, 77, 78, 176, 194,
 218, 245
 computational complexity, 19
 problem complexity, 90, 328, 331, 333

compromise, 18, 47, 57, 242, 252, 256, 264
conditions

 dominance conditions, 192, 213
 necessary condition, 212
 sufficient conditions, 203, 213
conflict, 181, 210, 242
constraint satisfaction, 213
convex cluster, 286

criterion, 18, 28, 46, 253

flow time, 86

machine loading ratio, 57

maximum completion time, 57, 87, 253

performance, 57

regular, 213, 243

robustness, 56, 80, 85

stability, 171

stochastic, 76, 101, 107, 143

tardiness, 253

critical chain, 26

critical path, 92, 102, 109, 180, 237, 281,
 292

D

decision aid, 237
discrete event system, 305
disjunctive problem, 204
dynamic algorithm, 24
dynamic programming, 78, 82

E

EDD rule, 80
enumeration, 185, 191, 216, 234, 304
exact
 evaluation, 235
method, 16, 256

- value, 147, 222, 255
- F**
 - flexibility, 20, 25, 27, 46, 255
 - assignment flexibility, 25, 199
 - sequential flexibility, 25, 199, 252, 320
 - temporal flexibility, 25, 200, 319
- G**
 - genetic algorithm, 124, 126, 254, 256
 - graph
 - conjunctive graph, 201, 233
 - Petri net, 306
 - precedence graph, 17, 93, 103, 173, 282, 284, 287
 - series-parallel graph, 109
 - time-bounds on node graph, 207
 - transitive graph, 255
 - graph, conjunctive-disjunctive graph, 252
 - greedy algorithm, 176, 187
- H**
 - heuristic, 157, 255, 281
- I**
 - interactive, 42, 237, 242
- J**
 - Johnson, 218
 - rule, 145, 218
 - sequence, 220
- L**
 - logistic chain, 265
- M**
 - makespan, 18, 58, 104, 254, 278
 - metaheuristic, 123, 155
 - multicriteria, 238
- N**
 - NP-complete, 79, 83, 101, 200, 327
 - NP-hard, 92, 128, 243, 254
- O**
 - on-line method, 20
- optimization
 - multicriteria, 18, 119
 - robust, 27, 30, 123, 124
 - stochastic, 29, 75
- P**
 - parallel application, 269, 272
 - parallel machine, 16, 54, 83, 87, 329
 - parametric analysis, 75
 - PARTITION, 332
 - permutation, 103, 128, 229, 265, 278, 280
 - flow-shop, 103, 143, 147
 - job, 218
 - sequences, 218
 - perturbation, 93, 195, 276, 278, 327, 340
 - Petri net, 305
 - precedence constraints, 17, 86, 133, 173, 309, 330
 - predictive, 24, 96, 171, 276
 - preemption, 17, 54
 - priority rule, 90, 135, 193, 236, 254, 255, 276
 - programming
 - constraint programming, 173
 - linear programming, 58, 59, 315
 - pyramidal structure, 211
- R**
 - RCPSp, 17, 171
 - stochastic, 181
 - release date, 18, 21, 127, 144, 212, 214, 328
 - resource allocation, 177, 178
 - robustness, 20, 26, 35, 44
 - conclusion, 44
 - measures, 27, 61, 129, 254
 - of a procedure, 44
 - of a set of solutions, 44
 - of a solution, 26, 28, 36, 44, 84
- S**
 - schedule
 - active schedule, 18, 252

- no-delay schedule, 18, 252
- proactive-reactive, 322
- semi-active schedule, 18, 251
- scheduling, 19
 - on-line, 25
 - predictive, 24
 - predictive-reactive, 265
 - proactive, 24, 57, 143, 171, 251, 256
 - proactive-reactive, 24, 199
 - reactive, 25, 237, 251, 260
 - stochastic scheduling, 99, 143
- sensitivity analysis, 31, 74
- service level, 109
- shop problem, 17, 54
 - flow-shop, 17, 103, 144, 217
 - job-shop, 17, 175
 - open shop, 17
- simulated annealing, 157
- simulation, 109, 112, 147, 181, 185
- single machine, 16, 78, 127, 204, 213
- slack, 177, 200, 236, 305
 - group slack, 238
 - proper slack, 238
 - sequential slack, 238
- SPT rule, 85
- stability, 28, 46, 178, 277
 - radius, 57
- T**
- tabu search, 123, 208
- U**
- unavailability, 37, 79, 324
- uncertainty, 20, 129
 - models, 22
 - sources, 21, 38, 143, 171, 172
- unit execution time, 331